Departamento de Ingeniería Telemática

PROYECTO FIN DE CARRERA

QRAPDID

Autor: Ezequiel Olivera Santos

Tutor: Mario Muñoz Organero

Leganés, “” de 2012

Título: QRAPDID
Autor: Ezequiel Olivera Santos
Director:

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Agradezco a Florina Almenárez, profesora del departamento de Ingeniería Telemática, por sus apuntes de J2ME.

Dar también agradecimientos a Pedro Lafuente por su ayuda y apoyo en parte del desarrollo de este proyecto al igual que a Julia Ruiz Martínez, gran aportadora de ideas de posibles alternativas de uso de esta aplicación.

Resumen

QRAPDID es una aplicación que consiste en la descodificación de un Código QR, el cuál contiene información de distinto carácter: vídeo, audio, texto o imágenes.

Esta aplicación persigue la comodidad de acceder a la información de forma rápida y sencilla con el simple gesto de realizar una foto a un código y extraer la información contenida en ella, con el fin de dar a conocer una información extra necesaria para que el usuario pueda interactuar de la manera más objetiva y segura, pudiendo elegir si desea o no conocer esta información. A parte de todo esto, se dispone de una aplicación web que permite tener reflejado el uso de QRAPDID por parte de los usuarios (esta aplicación es opcional para el Administrador del sistema puesto que en él se reflejan las estadísticas del uso de la aplicación) con el fin de "llevar las cuentas" del uso de esta aplicación dentro del entorno en el que se instale esta tecnología. Esta misma aplicación web permite crear Códigos QR mediante una sencilla interfaz en la que un selector y un área de texto es lo suficiente para que cualquiera pueda crear sus propios Códigos, los cuáles se usarán dentro del entorno donde se quiere utilizar QRAPDID.

Gracias a todo esto, QRAPDID se convierte en una herramienta potente de información alternativa donde los usos en distintos entornos es posible y el precio de su uso en mínimo, a parte de las ventajas que produce en ciertos entornos de tipo comercial.

Abstract

QRAPDID is an application that involves the decoding of a QR Code, which contains information of different kinds: video, audio, text or images.

This application seeks the convenience of accessing information quickly and easily with the simple act of taking a picture of a code and extract the informationcontained therein, in order to disclose extra information necessary for the user tointeract in the most objective and reliable, and can choose whether or not this information. Besides all this, there is a web application that allows the use ofQRAPDID reflected by the users (this application is optional for system administrator since it reflects the statistics of application usage) with to "takeaccount" of using this application within the environment in which to install the technology. This same web application to create QR codes through a simple interface in which a selector and a text area is enough for anyone to create their owncodes, which will be used within the environment where you want to use QRAPDID.

Thanks to all this, QRAPDID becomes a powerful tool which uses alternative information in different environments is possible and the price of its use in at least apart of the benefits it produces in certain commercial environments.

Índice

Capítulo 1: Introducción

A día de hoy el número de teléfonos móviles en el mundo es holgadamente grande; tanto es así que, en un estudio realizado en 2009, se estimó que el número de líneas móviles para 2010 rondarían los 4000 millones (parasaber.com) y que la tasa de penetración en algunos países superaría el 100%, o lo que es lo mismo, más móviles que personas censadas en un mismo país (como, por ejemplo, España). Ahora que estas predicciones se han convertido en hechos, el mercado del teléfono móvil quiere adquirir una expansión a nivel mundial (expansión por el continente Africano) debido a la gran utilidad del teléfono móvil y a su gran aceptación por parte de las personas, convirtiéndose en el dispositivo con el ritmo de adopción más rápido de la historia.

Esta gran aceptación (entre otras cosas) es lo que ha motivado a los fabricantes a evolucionar sus terminales móviles hasta tal punto que ya, hoy por hoy, hay incluso terminales con procesadores de doble núcleo (LG Optimus 2x). Pero no es lo único, si no que cosas tan simples como el poder reproducir mp3 con el móvil o hacer fotos con el mismo, entre otras cosas, es lo que le da el gran valor que el mismo tiene. Por lo que estas evoluciones y aumento de capacidades realizadas por los fabricantes permiten a los desarrolladores la posibilidad de crear aplicaciones potentes que permitan la interactuación del usuario con el móvil, el medio, etc... Todo ello gracias a la multitud de funcionalidades que poseen los móviles y que les hace convertirse en un "multidispositivo" eficaz, fiable y que permite gran variedad de acciones, como navegar por internet, usar el móvil a modo de linterna, mp3, reproductor de video, videocámara, etc... Poco más se puede decir del teléfono móvil, a nivel de usuario, que no se sepa ya. En nuestro país hay más de un teléfono móvil por persona y cada vez hay muchos más, por lo que el móvil es, claramente, un artilugio de la vida diaria al 100% y un elemento disponible al usuario las 24 horas del día (y no solo para llamadas o mensajes).

Por otro lado, se ha producido un fenómeno de gran éxito social en los últimos años debido a la gran ayuda que proporciona, la Realidad Aumentada (desde ahora RA). Cada vez más la tecnología sufre evoluciones increíbles, en un corto espacio de tiempo, cada vez menor. Uno de los fines más importantes que busca esta constante evolución es el de facilitarnos las cosas y hacernos la vida "un poco más fácil". Este mismo objetivo lo busca y lo consigue la RA ([6]), un artilugio que nos permite aumentar los conocimientos físicos mediante información adicional. La RA es el término que se usa para definir una visión directa o indirecta de un entorno físico del mundo real, cuyos elementos se combinan con elementos virtuales para la creación de una realidad mixta a tiempo real.. La diferencia con la realidad virtual es que no sustituye la realidad física, sino que sobreimprime los datos informáticos al mundo real. Con la ayuda de la tecnología (por ejemplo, añadiendo la visión por ordenador y reconocimiento de objetos) la información sobre el mundo real alrededor del usuario se convierte en interactiva y digital. La información artificial sobre el medio ambiente y los objetos pueden ser almacenados y recuperada como una capa de información en la parte superior de la visión del mundo real. A continuación vienen unos ejemplos de aplicación de RA:

Un ejemplo de RA es el sistema de BMW que guía a los mecánicos en el proceso de mantenimiento de los coches mediante unas gafas que proporcionan

información adicional de todos los elementos del coche, y sirve de ayuda para realizar reparaciones y poner/quitar elementos del mismo con una enorme fiabilidad. Es una ayuda importante la que proporciona este elemento que utiliza la RA para ayudar y mejorar un trabajo.

Como he dicho antes, consigue proporcionar un elemento de realidad más que ayuda y evita muchos problemas a sus usuarios (le facilita un poco más las cosas al usuario).

Otro ejemplo, esta vez para teléfonos móviles Android, es Enkin, una aplicación de visualización de la realidad que usa mapas en 3D de Google Earth y una cámara de video. En este caso, conectar los mapas virtuales con los espacios reales es el propósito de la RA, para dar más información de un edificio, un hotel... y nutrir así el conocimiento del usuario en esa situación en concreto.

Como hemos visto en estos ejemplos, la ayuda que da la RA es muy útil, y su finalidad es muy recomendada para cuando se necesita algún tipo de ayuda, de cualquier tipo, porque como vamos a ver más adelante, la RA se puede utilizar con multitud de ámbitos para conseguir el mismo fin, el de ayudar al usuario dentro del contexto en el que se encuentra.

QR-APDID une a un teléfono móvil y se basa en la RA (no es propiamente RA porque no muestra la realidad, sino que se muestra información relacionada pero sin sobreponerla a la realidad) para ayudar a los clientes. Es una aplicación didáctica con el propósito de informar, enseñar, ayudar o aconsejar mediante la utilización de una serie de tecnologías que permiten que la ocupación en el dispositivo sea mínima o progresiva, según lo que desee tanto el administrador como lo que permita el cliente o, por restricción, el terminal móvil del usuario. Su diseño se fundamenta básicamente en la intención de “mostrar el camino” ó “ayudar a encontrar el camino” dentro del ámbito donde lo ejecuta el administrador y permite a un usuario, con su terminal móvil, el aprendizaje o guía sin necesidad de que otra persona se lo cuente, simplemente mediante el uso de la cámara del móvil y la utilización de Códigos QR.

1.1 Motivaciones

Las motivaciones son varias, puesto que el uso de QR-APDID es aplicable a varios entornos donde el administrador puede crear esa especie de “mundo paralelo” gracias a la idea de hacer algo parecido a RA que esta aplicación proporciona. No tenemos que perder de vista la perspectiva de la RA aumentada ya que este es el método que nos permite obtener la información visual o auditiva (según el método de información utilizado en un momento determinado) y nos permite conocer todo lo que necesitamos saber para tomar una decisión. A partir de aquí se ofrecen una pequeña serie de circunstancias que motivan a la creación de QR-APDID:

Los seres humanos somos seres imperfectos y muy subjetivos. Esta teoría adquiere más fuerza cuanto más simple es la vara de medición puesto que no hace falta

hacer un estudio profundo y difícil para entender que los conocimientos de las personas a lo largo de la vida son su principal "marca", lo que hace a las personas. ¿Por qué no hace falta ir a cosas complejas y sí a cosas imples? Pues porque estas cosas simples son las que muestran (al igual que las complejas pero de manera más simple, valga la redundancia) los distintos que pueden ser las personas y no por ello significa que unas sean mejores que otras. Esto permite abrir un amplio abanico de gustos y posibilidades de elección en un ámbito concreto y posibilita, también, el posible interés de aprender más sobre ciertos temas en los cuáles las personas pueden creer que no tienen ningún dominio. La razón de toda esta explicación lleva a una sencilla pregunta: ¿Alguna vez se ha perdido en un centro comercial? A simple vista puede parecer una pregunta con no mucho sentido después de todo lo comentado pero, pregúnteselo, porque quizá encuentre una relación. Un centro comercial como, por ejemplo, IKEA (podría ser cualquier otro, pero este en concreto tiene unas peculiaridades bastante óptimas para el uso de QR-APDiD) tiene multitud de pasillos, multitud de camas, sofás, armarios, somieres, etc... Mi caso puede ser perfectamente el de una persona X que llega a dicho establecimiento con la intención de querer comprar algo pero no dispone de la suficiente experiencia, conocimientos o tiempo (en algunos casos puede ser el tiempo un factor importante para impedir la correcta comprensión de las cosas) como para realizar la compra perfecta o simplemente para comprar lo que verdaderamente necesite y la relación Calidad-Precio quede totalmente solventada. Los motivos de este desconocimiento no son la preocupación que nos interesa, pero sí nos interesa que X llegue y, haga lo que haga, quede contento con lo que ve, tenga una gran cantidad de información a su disposición para elegir y, en general, incentive su consumo, como el resto de personas que visiten el establecimiento. Sobre todo en estas épocas donde la crisis sigue siendo protagonista y en el sector comercial se traduce en menos personal para atender la demanda de los usuarios, puede ser conveniente el uso de QR-APDID ya que lo único que se necesitaría son: papel (para poder hacer la impresión de los Códigos QR) y que la persona en concreto disponga de esta aplicación en su móvil. Por lo que una persona que no tiene mucha idea acerca de qué es lo mejor para su nueva habitación, un sábado por la tarde (establecimiento abarrotado) llega a IKEA y sin necesidad de tener que buscar empleados del establecimiento (puesto que éstos, muy posiblemente, se encuentran ocupados) utiliza en el modo cámara de fotos para realizar capturas de los Códigos QR, los cuáles ofrecen toda la información necesaria acerca de un objeto en particular mediante distintos tipos de formas de ofrecer la información y, por supuesto, con el fin de aconsejar qué es lo mejor o lo peor del instrumento solicitado, con qué otras cosas se pueden combinar, relación Calidad-Precio, distintos lugares del establecimiento donde puede encontrar otros objetos del mismo fabricante u otros objetos combinativos con el solicitado, etc... Con esta utilización del móvil, estaríamos ante el nuevo código de "barras" personal, que nos ayuda en caso de no tener los conocimientos necesarios o, por lo menos, nos aconseja o nos da una opinión si somos personas conocedoras del ámbito. Con todo esto, QR-APDID se convierte en una gran ayuda a la hora de poder discernir entre cuáles son las cosas que más se pueden necesitar y cuáles son las mejores dentro de las que se necesitan (en diferentes ámbitos). Tampoco podemos olvidar las ventajas que esto le daría a establecimientos como el ejemplo que hemos puesto, aunque esto forma más parte de los objetivos buscados con esta aplicación, que los veremos más adelante, en el próximo apartado.

¿La motivación para hacer QR-APDID ha sido la de llegar a sitios "grandes" y no querer tener que esperar cola para preguntar, no saber qué puede ser necesario, etcétera? No, porque también están las pequeños tiendas de ropa (entendamos

por tiendas “pequeñas” aquéllas que no tienen tanta superficie como una nave donde pueda entrar un Carrefour, por ejemplo. Englobamos todo tamaño de tiendas únicamente de ropa). Es la misma escena que contábamos para centros comerciales, pero en distinto ámbito ya que ahora estamos hablando de ropa. Es un incordio cuando se llega a una tienda de ropa y se quiere hacer una serie de compras que, en algunos casos, no son para uno mismo (quizá se trate de un regalo para otra persona) y no se tiene la certeza de comprar lo apropiado; pues nunca viene mal una opinión externa, totalmente objetiva, no sesgada, acerca de una prende, unas zapatillas... y un consejo acerca de con qué se puede llevar puesto, con qué combina, etc... Como también dijimos antes, el interés es el de aconsejar y por eso también le puede servir a la chica que llega al local y que para comprarse una camiseta (aún sabiendo cuál quiere) utilice QR-APDID para corroborar la información que se le presta a modo de consejos. Por lo tanto, esta herramienta también puede ser muy eficaz para este tipo de situaciones en las que todo tipo de personas pueden obtener las explicaciones que QR-APDID da con, solamente, un par de pulsaciones o 3 en el terminal móvil.

Las motivaciones explicadas antes se deben a la problemática de la desinformación ó mala información, ya que permite al administrador (el administrador es, en realidad, el cliente o conjunto de clientes que disponen de la tecnología QR-APDID instalada en su negocio) dar información detallada del producto de forma alternativa a las formas tradicionales (que te lo cuente alguien) siempre que se cumplan dos requisitos fundamentales: que el cliente tenga esta aplicación y que el cliente quiera obtener más información del producto (más adelante explicamos objetivos).

Para finalizar con este punto, podemos hablar de otro caso en el que QR-APDID puede ser una gran ayuda: en espacios abiertos (un campus universitario, un bosque, una montaña...). A modo de indicadores y teniendo en cuenta que, como dijimos antes, todo el mundo dispone de terminal móvil, QR-APDID puede ser la mejor herramienta para llegar a un punto concreto dentro de espacios amplios donde pueda haber un amplio margen de error para encontrar el destino deseado. Solo habría que poner “señales” en forma de Códigos QR para que la persona haga una foto (misma temática que en los casos anteriores) y obtenga información acerca de su posicionamiento, sitios de interés cercanos, destinos posibles, etc. Esto aumentaría todos los conocimientos acerca del terreno por parte de la persona en cuestión y todo ello sin necesidad de llevar un GPS encima (con todo lo que ello acarrea) o, por otra parte, podría complementar el servicio dado por el GPS mediante textos o vídeos donde se expliquen todos los aspectos importantes que se deben tener en cuenta en el punto donde se encuentre la persona, informaciones que posiblemente ni un GPS puede dar (peligros cercanos, advertencias, etc...). Todo esto está pensado para montañeros de todo ámbito (ciclistas, personas que van a pasar el día a la montaña, corredores...) o para alumnos que desconocen su campus (Campus de Moncloa de la Universidad Complutense de Madrid) ya que no está de más tener información en cualquier sitio sin necesidad de tener que buscar a nadie o tener que esperar a encontrarlo de manera aleatoria o cansada.

Estos tipos de sitios pueden ser inicios de otros más donde se pueda implantar esta tecnología, debido a su carácter alternativo y opcional, puesto que el cliente puede obtener más información, o no (si él no quiere), pero siempre será una manera de completar todos los conocimientos que una persona puede tener acerca del ámbito en concreto o incluso una manera de aprender acerca del mismo. Este es el

motivo por el cual la aplicación se denomina QR-APDID, Aplicación DIDáctica que utiliza Códigos QR. El nombre no es muy original ni rebuscado, pero define perfectamente el funcionamiento de esta aplicación y permite establecer una clara comprensión de los objetivos que busca la misma. Por ello, QR-APDID es una sencilla aplicación didáctica capaz de ofrecer conocimientos objetivos y consejos acerca de numerosos temas determinados dentro de un ámbito previamente creado e instaurado por un administrador.

1.2 Objetivos

Desde luego que nunca es la intención suplantar la "ayuda humana", es decir, que QR-APDID no quiere incentivar a las empresas a eliminar personal (aunque tampoco es una obligación), sino que lo que se quiere conseguir es que haya más información al alcance de los clientes sin tener que realizar esperas o tener que investigar por su propia cuenta. Esta máxima, que es entendible desde el ámbito comercial, ofrece a los administradores una nueva forma, una nueva alternativa, de incentivar el consumo, más ahora en épocas en las que el consumo se ve seriamente afectado por la crisis económica y el mismo se puede ver más afectado por motivos tan simples como la menor atención que se le pueda prestar a los clientes por parte de los trabajadores de la empresa, teniendo en cuenta que ahora el número de los mismos es menor que en épocas de mejor nivel económico. Por todo esto, QR-APDID se traduce en una herramienta con gran capacidad de información específica que reportará más beneficios a las empresas por lo comentado antes, ya que mientras más información de los producto tengan los clientes, más capacitados estarán los mismos para realizar compras y, en definitiva, consumir. Como hemos apuntado antes, QR-APDID no busca la suplantación, sino más bien el apoyo, un apoyo que sirve de base para difundir, de forma objetiva, toda la información que los propios trabajadores no tendrían que saber acerca del producto (o, por lo menos, no a gran escala) de manera objetiva y con el fin de informar pero, sin embargo, también puede ser una herramienta que permita que la mano de obra disminuya sin verse afectado el servicio prestado por el establecimiento. Así que QR-APDID se puede ver como herramienta de apoyo pero también se podría ver como herramienta de ahorro, un ahorro de mano de obra que también reportará beneficios a la empresa en cuestión puesto que es mano de obra de menos que tiene que pagar. Todos estos fines se ven desde el punto de vista del administrador que es la persona o conjunto de personas que se encargan de elaborar un criterio de códigos y una serie de estándares de mensajes que, posteriormente, serán leídos por los clientes con solo hacer un "clic" en sus terminales móviles. Pero, ¿qué aporta a los clientes? A los clientes aporta un conocimiento más exacto, una nueva forma de concebir los datos necesarios para poder discernir acerca de los productos, un mecanismo sencillo que evita esperas, que interviene como "ayudante" a la hora de realizar la compra, o que, de forma opcional, aconseja a los clientes, puesto que no hay que perder de vista de que este mecanismo puede ser usado de forma totalmente opcional por parte de los clientes (como bien dijimos en apartados anteriores, dos cosas han de cumplirse: el cliente debe disponer de QR-APDID en su terminal móvil y el cliente puede elegir entre usar o no dicha herramienta en el escenario donde se encuentra habilitado ya que este es un mecanismo puramente opcional). Todo esto ayuda al cliente, tanto al que sabe como al

que no: al que sabe, porque aún sabiendo tiene la posibilidad de preguntar a un empleado y/o utilizar QR-APDID para verificar, conocer más a fondo o vetar las características de un objeto concreto; mientras que para el que no sabe, toda esta información se convierte en una mina de conocimientos que permitirán al cliente tomar decisiones a partir de una base óptima, objetiva y segura. Y, por supuesto, también aporta ahorro a los clientes, pero este es un ahorro de tipo temporal, del que ya hemos hablado pero no hemos dado la suficiente importancia, un ahorro de tiempo a la hora de obtener la información necesaria de los productos en los que está interesado.

Todas estas afirmaciones están vistas en el contexto de QR-APDID como herramienta comercial capaz de elaborar un entorno en el que los clientes puedan interactuar con los Códigos QR para adquirir la información. Esta información puede estar representada de varias formas y la fuente puede ser distinta. Siempre será el administrador el que elija cuál o cuáles son las formas en que desea gestionar, guardar y mostrar la información que desea servir a sus usuarios. Por ello QR-APDID también busca que las personas que quieran trabajar con esta herramienta lo puedan hacer con pocas restricciones (De todos modos, más adelante explicaremos cuáles son estas formas de servir, guardar y gestionar la información, ya que esto no forma parte de este punto). Por ello, podemos decir que QR-APDID busca la "adaptabilidad", entendiéndola como la capacidad de adaptarse a los requisitos que el administrador ponga o a las restricciones que el mismo tenga.

QR-APDID busca la sencillez, ya que en un par de "clics" ya puede la persona disponer de toda la información necesaria tras haber hecho una simple foto a un Código QR, por lo que esta aplicación puede ser bastante bien acogida por el público joven, adulto y anciano debido a su fácil funcionamiento y sencilla utilización.

Después de todas estas finalidades que esta aplicación busca (sencillez, adaptabilidad, apoyo, ahorro...) hay que hacer un importante hincapié en la labor de orientación que, evidentemente, no es la misma dentro de los contextos en los que esta herramienta puede ser utilizada. Por ejemplo, la orientación que se busca en el contexto comercial es la de orientar al cliente a elegir el producto que más se corresponda con los expectativas que el cliente busca mientras que, por otra parte, la orientación que se busca dentro de un espacio abierto (en las inmediaciones de una montaña, por ejemplo) corresponde a la motivación a ayudar a la persona a encontrar un camino determinado por el que seguir o informar acerca de los posibles destinos que puede escoger para explorar o visitar. Así que QR-APDID también trata de motivar a las personas para que las mismas puedan escoger entre disantos destinos para explorar mediante la orientación, a la vez que intenta guiar a los mismos ante posibles pérdidas dentro de un ámbito de lugar grande y abierto, consiguiendo dar una gran capacidad de información que las señales convencionales no son capaces de dar puesto que estas últimas solo dan una información exacta y concreta, mientras que QR-APDID no solo da esta información exacta, si no que la complementa con datos más concretos acerca del lugar donde se encuentran las personas (como antes apuntamos, incluso se puede ver como un apoyo al sistema GPS como forma de adquirir más información en el punto donde se encuentra la persona, o como sustituto del mismo, puesto que no existe ninguna máxima que obligue a llevar un GPS las 24 horas del día a cualquier sitio que una persona vaya).

Todos estos son los objetivos fundamentales que esta aplicación busca, ya que lo interesante de todo esto es la posibilidad de disponer de una información

opcional capaz de clarificar todo lo posible las dudas que se puedan tener acerca de los temas que reinen dentro de un ámbito concreto. Por ello, QR-APDID une sencillez, adaptabilidad y fiabilidad para que tanto clientes como administradores puedan desarrollar un escenario específico donde lo más importante es la información, ya que con ella todas las decisiones tomadas serán a partir de unos conocimientos previos que permitirán multitud de ventajas tanto a unos como a otros. Todo esto lleva a comentar que esta aplicación, en principio, no tenía un carácter comercial, pero si se mira desde el punto de vista del negocio, se puede ver que la ayuda que aportaría es de un valor posiblemente incalculable.

Para finalizar con este tema, a continuación veremos un artículo ([12]) del 22 de agosto de 2011, donde nos explican un montón de ideas acerca de cómo utilizar un CQR (una motivación clave para utilizar este tipo de código bidimensional):

"Durante el año pasado, el entusiasmo acerca de los **códigos QR** se ha convertido aparentemente en una base diaria. Hemos visto cada vez más empresas utilizando este código (en España todavía no) para promover su empresa, productos o servicios. Hemos visto a bibliotecas y escuelas ponerlas en práctica para ayudar a los estudiantes y a los padres de familia. También hemos visto que proporcionan valor a mercados verticales específicos, tales como bienes raíces y de salud.

Hemos compilado una lista de 100 maneras que usted podría usar los **códigos QR**. Hemos recibido algunas de estas ideas de los clientes, de Internet y amigos. Algunos de ellos provienen de nuestras propias experiencias. Esperamos que usted encuentre al menos algunas de estas útiles en algunos de estos ejemplos que vienen a continuación:

- Póngalos en señales, para que la gente puede seguir las rutas de autobuses y los tiempos de llegada.
- Agregar ellos en / cerca de los monumentos históricos, los dirigen a los vídeos con más información.
- Póngalos en la propiedad de los vehículos de la ciudad para que la gente pueda ver lo que cuesta y lo rápido que se deprecia.
- En los "Automóviles" de una revista de prensa / folleto /, poner un **código QR** para que la gente vea los Automóviles de la Semana.
- Añadir un **código QR** en los carteles promocionales permiten a la gente realizar el "me gusta" en las páginas de Facebook.
- Poner **códigos QR** en la ropa (camisetas) para llevar a la gente a su página de Twitter.
- En las fotos, poner los **códigos QR** que conducen a las personas a los perfiles deLinkedIn de la gente en las fotos.
- En la sección de Bienes Raíces de una revista de prensa / folleto /, poner un **código QR**parabque la gente vea la Casa de la Semana.
- En bienes raíces, añadir un **código QR** para ver los pisos piloto. Para que la gente vea en directo un vídeo / tour virtual de la casa.
- Añadir un **código QR** en su tarjeta de visita que ofrece a las personas su información de contacto.
- Añadir un **código QR** en su tarjeta de negocios que dirige a la gente al sitio en línea – puede contener un vídeo de usted o de su empresa, enlaces a los perfiles de los medios de comunicación social, y a su sitio Web corporativo.

- Añadir **códigos QR** a las distintas secciones de un libro que la gente vea directamente aumentar el contenido en la Web.
- Pon un **código QR** junto a columnistas en un periódico. Se puede dirigir a la gente a una página en línea en las listas de otros artículos de ese escritor.
- **En los restaurantes**, poner los **códigos QR** en los tableros. Estos pueden dirigir a las personas a las páginas de los medios de comunicación social del restaurante.
- En las tiendas, añadir los **códigos QR** en los escaparates, cajas registradoras y otros signos. Estos pueden ordenar las páginas optimizadas para móviles, donde pueden registrarse en su boletín de noticias.
- En los folletos de su empresa, añadir los **códigos QR** para que la gente vaya directamente a los vídeos, podcasts y páginas Web que seguir debatiendo el tema correspondiente de la pieza impresa.
- Poner **códigos QR** en las piezas de correo directo para que a la gente se le conduzca a URLs personalizadas.

Estas y más ideas hacen que los **códigos QR** sean cada vez mayormente utilizados por las personas".

Con todo esto, ahora vamos a pasar a describir las tecnologías utilizadas para la creación y puesta en marcha de QR-APDID en el siguiente capítulo, que es el Estado del Arte, en el que se hará una exhaustiva explicación a todo detalle de todas las tecnologías con el fin de desgranar todos los entresijos necesarios para la creación de esta aplicación.

Capítulo 2: Estado del Arte

Vamos a hacer una categorización de las distintas tecnologías utilizadas de modo que se ubicarán todos los aspectos relevantes y explicaciones en diferentes apartados. Evidentemente, cada apartado corresponde a una tecnología en concreto. También se hará una introducción a las mismas para saber en qué contextos nos encontramos.

2.1 Códigos Bidimensionales (Códigos QR)

Atendiendo al trabajo que vamos a desarrollar, para la creación de QR-APDID vamos a utilizar la plataforma Java J2ME, una plataforma en lenguaje Java para la creación de aplicaciones móviles para dispositivos móviles cuya descripción detallada de la misma la haremos más adelante, y vamos a usar los Códigos Bidimensionales, cuya definición, tipos de códigos, ventajas, etc., las vamos a mostrar a continuación para saber con qué vamos a trabajar.

2.1.1 Introducción

Los Códigos Bidimensionales ([1] y [2]) son iconos o símbolos formados por una combinación de puntos y barras. Constan de una matriz que permite el escaneo rápido de la información que contiene, de tipo omnidireccional, obteniendo su decodificación a alta velocidad. Además, permiten aumentar considerablemente el número de caracteres codificados en un solo código con respecto a los lineales, pasando de 20 a más de 7000. De esta manera se pueden representar todo tipo de caracteres numéricos y alfanuméricos: Kanji, Kana Hiragana, símbolos binarios y códigos de control.

Existen muchos tipos de códigos bidimensionales, como son los códigos **QR** (*Quick Response Barcode*), que son los que vamos a utilizar y los que definiremos en los siguientes apartados, **DataMatrix**, **Maxicode**, **Aztec code**, **PDF417**, **RSS** (*Reduced Space Symbology*) o los **códigos 49**. Estos códigos pueden contener información de todo tipo, desde información adicional de un producto (descuentos, precios, cómo llegar a la tienda, etc.), hasta mensajes ocultos, juegos de pistas, teléfonos, direcciones de e-mail, enlaces web (descargas de juegos, canciones, etc.), SMS, o incluso las coordenadas sobre un lugar geográfico concreto o el tiempo que va a tardar en llegar el autobús a una determinada parada (estos datos han sido sacados de “Wikipedia, la enciclopedia libre”).

2.1.2 Ventajas de su uso

Entre las ventajas que presentan los códigos bidimensionales ([2]) cabe destacar que son muy fáciles y baratos de implementar, pues basta con imprimirlos en pegatinas, carteles o directamente en el envase del producto. Luego basta con hacer una foto al código con el móvil o PDA para que la información aparezca directamente en la pantalla. Además la información se puede recuperar fácilmente, incluso aunque la etiqueta haya sido dañada. Otra ventaja es que, como dijimos antes, los códigos bidimensionales pueden hacer la misma función que los códigos lineales utilizando significativamente menos espacio. Códigos lineales como el 39 ó el UPC sólo pueden codificar entre 10 y 20 caracteres la mayoría de las veces, los códigos bidimensionales pueden codificar varios miles de caracteres legibles por una máquina. Los códigos bidimensionales son mucho más resistentes a daños que los lineales gracias a fórmulas de corrección de errores que utilizan. Algunos códigos pueden perder hasta un tercio de su superficie y aún ser decodificados. El hecho de que cada vez existan más terminales móviles capaces de leer éste tipo de códigos, y que los lectores necesarios sean gratuitos, está provocando que los códigos bidimensionales sean el método cada vez más empleado para el marketing móvil. Esto se debe también a que permiten una interacción directa y confidencial con el público, permitiendo que los clientes disfruten de numerosos servicios de forma inmediata y sencilla.

2.1.3 Ejemplos de su uso

Uno de los primeros usos que se está dando a los códigos bidimensionales en España es en la venta de **billetes de autobús** y **entradas de cine**. El usuario recibe en el móvil el código correspondiente a su compra, y lo presenta como comprobante en la estación o el cine, sin tener que imprimir tickets ni aguardar colas.

En Japón, donde el uso de estos códigos está mucho más extendido, gran número de productos llevan **impreso en su envoltorio** un enlace a la página web de su fabricante, donde puede encontrarse información añadida. Esto ha incitado a diseñar códigos bidimensionales más atractivos que aumenten el número de clientes potenciales, dándoles **forma circular**, incorporando **imágenes estáticas** en el código, como puede ser el logo de la compañía, o incluso creando **vídeos** con ellos.

2.1.4 QR (Quick Response Barcode)

Entre los códigos bidimensionales existentes, vamos a usar los Códigos QR (a partir de ahora CQR). Un Código QR (Quick Response Barcode) es un sistema para almacenar información en una matriz de puntos o un código de barras bidimensional creado por la compañía japonesa Denso-Wave en 1994; se caracterizan por los tres cuadrados que se encuentran en las esquinas y que permiten detectar la posición del código al lector. La sigla "QR" se derivó de la frase inglesa "Quick Response" pues el creador aspiraba a que el código permitiera que su contenido se leyera a alta velocidad. Los códigos QR son muy comunes en Japón y de hecho son el código bidimensional más popular en ese país. Las características generales de los CQR son las siguientes:
Aunque inicialmente se usó para registrar repuestos en el área de la fabricación de vehículos, hoy, los CQR se usan para administración de inventarios en una gran variedad de industrias. Recientemente, la inclusión de software que lee CQR en teléfonos móviles japoneses, ha permitido nuevos usos orientados al consumidor, que se manifiestan en comodidades como el dejar de tener que introducir datos de forma manual en los teléfonos. Las direcciones y los URLs se están volviendo cada vez más comunes en revistas y anuncios japoneses. El agregado de CQR en tarjetas de presentación también se está haciendo común, simplificando en gran medida la tarea de introducir detalles individuales de un nuevo cliente en la agenda de un teléfono móvil.

Los consumidores que cuenten con dispositivos y programas de captura, en combinación con un PC con interfaz RS-232C pueden usar un escáner para leer los datos.

El estándar japonés para CQR (JIS X 0510) fue publicado en enero de 1999 y su correspondiente estándar internacional ISO (ISO/IEC18004) fue aprobado en junio de 2000.

Un detalle muy importante sobre el código QR es que su código es abierto y que sus derechos de patente (propiedad de Denso Wave) no son ejercidos ([1]).

2.1.5 Formato y almacenamiento de CQR

La forma en que se almacenan los datos en un CQR es la mostrada a continuación:

Arquitectura Código QR

Como podemos comprobar, el CQR es muy semejante a otros códigos en el sentido de ser un conjunto de barras y puntos capaces de almacenar información de distintos tipos. Este tiene unas curiosidades ya explicadas en su forma y es capaz de manejar y guardar un gran número de datos en su seno.

2.1.6 Ejemplos de uso de CQR

Anuncio Mini - CQR

Vidriera con CQR

Fachada con CQR

Desde anuncios de coches hasta apariciones en fachadas de importantes compañías, el código QR es una barata y potente solución a la adquisición de información a partir de los mismos.

Al código QR también se le conoce como BIDI, un sistema muy semejante que, de hecho, utiliza código QR. Por ello son equivalentes y se puede encontrar tanto un nombre como otro para referirse a la misma tecnología.

2.1.7 Soporte para lectura de Códigos Bidimensionales en Java J2ME (Códigos QR o BIDI)

Hemos realizado un estudio más en profundidad de este código debido a que muchos móviles con cámara, cuyas aplicaciones son ejecutadas en Java, pueden disponer de un fácil y sencillo programa capaz de desencriptar estos formatos de código y además porque es la clase de código bidimensional con el que se puede trabajar en la

tecnología J2ME. Hay bastantes lectores, incluso más de uno para un modelo móvil concreto, o un lector que vale para más de un modelo.

Hay códigos ya implementados de programas en Java que permiten tomar fotos de códigos QR y poder descifrar su contenido (evidentemente, existe el código si existe la aplicación). Con lo cual aquí se unen dos grandes grupos de estudio, el del soporte para lectura de J2ME, y el de soporte de multimedia para dicho lenguaje mediante las capturas de cámaras.

Aun así, hay que saber que para permitir el uso de herramientas capaces de descodificar un código QR, el programa debe importar una serie de librerías creadas para el uso y lectura de códigos QR: Primero tenemos una librería de excepciones para lanzar las mismas cuando no se pueda leer algo, o no sea el formato esperado, o no se pueda decodificar el objeto (Clases que extienden de Excepción y tienen el mismo "cuerpo" que clases cuyo fin es el mismo o parecido); también habrá que soportar una interfaz que escenifica la imagen (QRCodeImage) y una clase completa de la que se puede sacar el formato, versión, longitud, y multitud de cosas del símbolo (QRCodeSymbol), que utiliza un código cíclico llamado BCH15 5, aparte de aparecer la matriz de datos y acciones sobre la misma; se usa también la definición del punto (clase Point) con sus coordenadas x e y (bidimensional) y acciones sobre el mismo como la comparación con otro, distancia hacia otro, etc.; se crea una clase, "QRDataBlockReader", que lee los datos del símbolo leído, además, se pueden leer en distintos formatos.

Para proceder a la decodificación de un símbolo QR, se usan estas clases procedentes de librerías ya definidas que usan todo el código que les permite usar tanto CLDC 1.0 como MIDP 2.0 (de hecho, en CLDC 1.0 no existe la clase Math, en cuyo caso se tiene que predefinir manualmente, como se hace en el ejemplo de esta aplicación) y que junto a definiciones hechas de decodificación de código y manipulación del mismo, más la forma de conseguir los datos, que lo veremos más adelante, podemos reproducir un lector de códigos QR fiable y bueno que nos permitirá leer estos códigos y conseguir el objetivo buscado.

Después de haber visto cómo dar soporte de lectura de códigos a una aplicación java mediante la utilización de una serie de librerías (de forma bastante sintetizada), a continuación, vamos a analizar los aspectos más importantes de la tecnología J2ME y, posteriormente, vamos a ver el tratamiento que da Java J2ME a la multimedia.

2.2 J2ME

La realización del proyecto se hará con la plataforma **Java Micro Edition**, o anteriormente **Java 2 Micro Edition** (**J2ME**), es una especificación de un subconjunto de la plataforma Java orientada a proveer una colección certificada de APIs de desarrollo de software para dispositivos con recursos restringidos. Está orientado a productos de consumo como PDAs, teléfonos móviles o electrodomésticos.

Java ME se ha convertido en una buena opción para crear juegos en teléfonos móviles debido a que se puede emular en un PC durante la fase de desarrollo y luego subirlos fácilmente al teléfono. Al utilizar tecnologías Java el desarrollo de aplicaciones o videojuegos con estas APIs resulta bastante económico de portar a otros dispositivos.

Esta plataforma ha sido diseñada por Sun MicroSystems, y los dispositivos que soportan Java ME implementan un perfil. El perfil más usual en dichos dispositivos es el **Mobile Information Device Profile (MIDP),** para dispositivos tales como teléfonos móviles. Los perfiles son un subconjunto de las configuraciones, que se dividen en dos grupos actualmente: el **Connected Limited Device Configuration (CLDC)** y el **Connected Device Configuration (CDC)**. A nosotros, de estas configuraciones, nos interesa solo la CLDC, porque es la configuración orientada a dispositivos limitados (teléfono móvil, por ejemplo), conteniendo un estricto subconjunto de librerías de clases Java y es la cantidad mínima necesaria para que opere la Java Virtual Machine. Al final, todo se convierte en un sistema de niveles (máquina virtual, configuración, perfil) que representa la arquitectura J2ME y permite desarrollar aplicaciones en lenguaje Java para dispositivos móviles (por encima de los perfiles existen los paquetes opcionales que, como indican su propio nombre, se pueden utilizar opcionalmente), por eso no nos interesa la configuración CDC, porque consiste en un subconjunto de Java SE (Standard Edition), con un montón de librerías que no son relacionados con dispositivos móviles, por ejemplo y, por tanto, es una configuración mucho más rica que la CLDC. Así pues, como decíamos antes, el perfil a utilizar será MIDP, un perfil diseñado para la realización de aplicaciones para teléfonos móviles que incluye una GUI (Interfaz Gráfica de Usuario) y una fuerte API para datos. La versión 2.0 contiene, además, un fuerte API de juegos 2D. Las aplicaciones son llamadas “MIDlets”. Como estas características hacen a J2ME un lenguaje de programación de dispositivos móviles muy fuerte (sobre todo cuando se utiliza como perfil MIDP 2.0), muchos móviles de la actualidad vienen con una implementación MIDP y se ha convertido en el estándar actual de juegos móviles descargables. Sin embargo, muchos teléfonos móviles pueden arrancar esos MIDlets que han sido aprobados por la “empresa portadora” de esta tecnología, sobretodo en Norte América. En la actualidad, se ha desarrollando MIDP 3.0 (MIDP3), expandiendo la funcionalidad en todas las áreas tanto como mejorando la interoperabilidad entre dispositivos ([3] y [4]).

2.2.1 Resumen de Arquitectura J2ME

Desde más abajo hasta más arriba:

Máquina Virtual: Interpreta código intermedio java, efectúa llamadas al SO, etc.:

- CVM: Mejor uso de memoria, procesadores de 32 bits, ligada a config CDC
- KVM: dispositivos más limitados, ligada a la config CLDC

Configuraciones: Mínimo conjunto de clases disponibles para un grupo de dispositivos. Los grupos se establecen según requisitos similares de memoria y procesamiento. Defina las características soportadas del lenguaje java y la JVM:

- CDC: Connected Device Configuration.
  - Orientado a dispositivos con 512 KB de ROM y 256 KB de RAM
  - Conexión a red (fija)
  - Soporte completo a las especificaciones de JVM
- CLDC: Connected Limited Device Configuration.
  - Orientado a dispositivos con 150-512 KB de memoria disponible para Java.
  - Procesador de 16-32 bits, velocidad de 8-32MHz
  - Limitaciones de consumo (baterías)
  - Conectividad a red (inalámbrica)
  - Restricciones importantes en el interfaz de usuario

Perfiles: conjunto de clases Java que complementan una configuración para un conjunto específico de dispositivos. Definen las APIs que controlan el ciclo de vida de la aplicación, la interfaz de usuario, etc. Permiten portabilidad de aplicaciones entre diferentes dispositivos. (Ejemplo: MIDP)

Paquetes opcionales: Conjunto de APIs adicionales que pueden ser añadidos de forma flexible sobre diferentes perfiles. Un dispositivo puede soportar múltiples paquetes opcionales. (Ejemplo: MMAPI)

2.2.2 ¿Qué engloban los niveles?

CLDC/KVM:

→ **Cubre**	→ **NO cubre**
- Máquina virtual y soporte a Java	- Gestión del ciclo de vida de las aplicaciones
- Modelo de Seguridad	- Interfaz de usuario
- Entrada/Salida	- Gestión de eventos
- Soporte a conexiones de red	- Modelo de aplicación a alto nivel
- Internalización	- Soporte a almacenamiento persistente

- Internacionalización: todos los dispositivos soportan por defecto `ISO-LATIN1`
- Propiedades: la llamada a `System.getProperty(String key)` devuelve el valor de la propiedad.
- E/S a sistemas de almacenamiento y red: *Generic Connection Framework*
- Seguridad: no soporta el modelo completo de J2SE. (Utiliza verificador de clases y modelo sandbox)

MIDP:

→ **Cubre**
- Ciclo de vida de la app - Interfaz de usuario - Soporte de red
- Almacenamiento persistente

→ **NO cubre**
- Descarga y gestión de apps en los moviles - Seguridad a bajo nivel
- Gestión de baterías, codificadores de voz…

- Se asume la existencia de *Application Management System* (**AMS**).
  - Dependiente del dispositivo
  - Instala, interacciona con y borra midlets.
- MIDP debe proporcionar al menos las propiedades `microedition.locale` y `.profiles`
- Librerías:
  - Interfaz de usuario: `javax.microedition.lcdui` `(lcdui.game)`
  - Ciclo de vida de la app: `javax.microedition.midlet`
  - Memoria persistente: `javax.microedition.rms`
  - Conectividad: `javax.microedition.io`
  - Núcleo: `javax.microedition.io .lang` `.util`
  - Seguridad (clave publica): `javax.microedition.pki`
  - Sonidos: `javax.microedition.media` `(media.control)`

2.2.3 Desarrollo de aplicaciones (“MIDlets”)

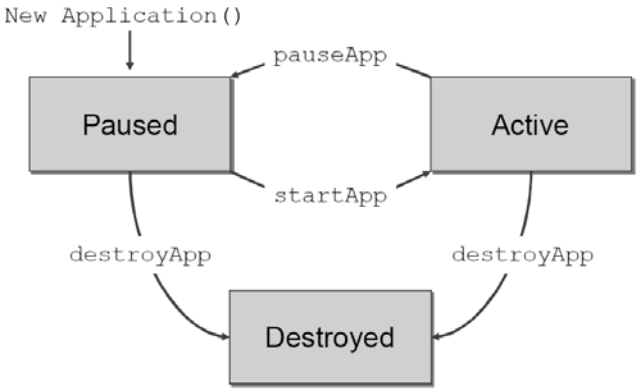
1. Creación: escribir código y compilar, preverificar código, JAR y JAD, ejecutar, depurar.
2. Publicación: en un servidor, por ejemplo.
3. Descarga: gestionada por AMS. Negociación de capacidades…
4. Instalación: gestionada por AMS. Puede transformar el midlet a un formato local.
5. Ejecución: el midlet entra en la VM y se invocan los métodos que gestionan su ciclo de vida.
6. Actualización: gestionada por AMS.
7. Borrado: el AMS debe permitir al usuario eliminar midlets.

- JAR y manifiesto: Incluye los ficheros de clases y otros recursos asociados al midlet. El manifiesto está incluido en el JAR y contiene información sobre los contenidos del JAR.

- Descriptor JAD: Permite que el AMS verifique antes de descargarlo si el midlet es adecuado. Es un fichero de texto con extensión .jad.
    - Atributos obligatorios: MIDlet- Name, Version, Vendor, Jar-URL, Jar-Size

2.2.4 Ciclo de vida de un MIDlet

Ciclo de vida de un MIDlet



Una vez conocidos los rasgos más importantes de toda la arquitectura J2ME para la realización de aplicaciones móviles y tras haber visto un esbozo de cómo darle a nuestro MIDlet el soporte para la lectura de CQR, ahora nos vamos a detener en el estudio del soporte de multimedia en J2ME, ya que este es un bloque importante en la creación y desarrollo de QR-APDID, pues es lo que nos da soporte para la utilización de la cámara.

2.2.5 Soporte de Multimedia (en J2ME)

Antes de centrarnos en la parte que más nos interesa del tratamiento de Multimedia por parte de la tecnología J2ME, que es la parte que tiene que ver con la captura de cámara, vamos a ver un ligero resumen de la gestión de Multimedia por parte de dicha tecnología:

Paquete Opcional (JSR 135) que proporciona funcionalidad multimedia genérica. Son un conjunto de objetos que permiten el tratamiento de la multimedia con la orientación a objetos definitoria de Java (Las acciones que se esperan son la generación, reproducción y grabación de audio o vídeo).

3 paquetes:

```
javax.microedition.media
javax.microedition.media.control
javax.microedition.media.protocol
```

Las características principales de MMAPI son: MMAPI puede ejecutarse en cualquier Virtual Machine y configuración (CDC, CLDC), facilita el incluir determinada funcionalidad, es extensible (pueden añadirse nuevos características o formatos) y permite un fácil manejo de los objetos que permiten tener multimedia.

Los componentes (Objetos que permiten la funcionalidad) de MMAPI son los siguientes:

DataSource

– Encapsula y abstrae la gestión del protocolo, ocultando los detalles de cómo se leen los datos de la fuente.
– Permite al objeto **Player** gestionar el contenido.

Manager

– Permite crear Player a partir de DataSource

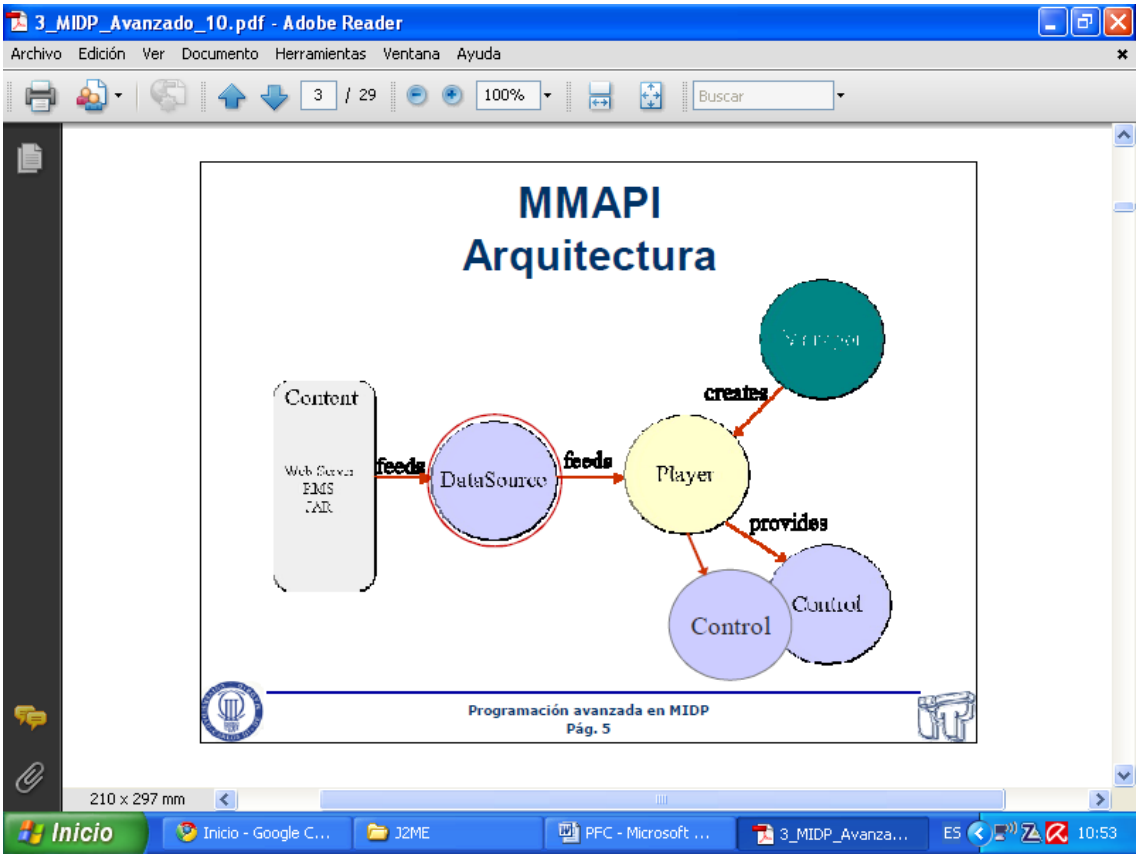
Player

– Lee los datos de DataSource y los procesa.
– Los manda al dispositivo de salida.
– Incluye métodos para el control de la reproducción.

Contro
l

– Controla las características de un Player y operaciones de reproducción

A continuación, un esquema donde vemos cómo es la arquitectura de MMAPI:



En este esquema vemos de manera práctica la relación entre los distintos objetos que nos permiten la gestión de la multimedia en J2ME. A continuación explicamos cómo actúan estos objetos.

- Las clases de MMAPI:

La clase **Manager** es el punto de partida para procesar o reproducir datos multimedia. A partir de él ocurre lo siguiente:

– Creación de objetos Player.

– Reproducción de tonos simples:

• **playTone(int nota, int duracion, int volumen)**

Clase factorial de Players:

– **Player player = Manager.createPlayer(String url);**

– URL: **<protocol>:<content location>**

– La aplicación usa los métodos del Player creado para controlar la descarga y reproducción de datos multimedia

Consultar propiedades:

– getSupportedContentTypes()

– getSupportedProtocols()

Estados de **Player:** El ciclo de vida de un **Player** incluye 5 estados, y 6 métodos que controlan las transiciones entre estados. Hay que tenerlos en cuenta a la hora de usar los objetos **Player.**

Con los objetos de tipo **Player** podemos proceder a la captura de evento para el uso de multimedia. Los eventos de **Player** deben seguir las siguientes pautas:

-Un objeto **PlayerListener** debe implementar el método playerUpdate()
-La instancia de la interfaz se asigna al objeto **Player**
`addPlayerListener()`
- Eventos predefinidos notificados en el objeto **Player**
`END_OF_MEDIA, ERROR, DURATION_UPDATED, DEVICE_AVAILABLE ...`

Un **Player** proporciona control específico para el tipo de medio que procesa:
– **getControl()** para un único control.
– **getControls()** para un array de controles.
– Ejemplo: Un player de MIDI que invoque **getControl()** recibe un **MIDIControl**.

Define 12 controles:

– FramePositionControl	─ GUIControl
– MetaDataControl	─ MIDIControl
– PitchControl	─ RateControl
– RecordControl	─ StopTimeControl
– TempoControl	─ **ToneControl**
– VideoControl	─ **VolumeControl**

Con todo esto ya podemos acceder a la captura de imágenes a través de la utilización de una cámara de un dispositivo móvil, por lo tanto, J2ME nos permite tener todos los elementos necesarios para conseguir idear una aplicación capaz de utilizar códigos QR y poderlos descodificar mediante un protocolo de captación de la información que se explicará más adelante (ya tenemos lo mínimo indispensable para utilizar QR-APDID).

En el siguiente apartado, como ya hemos explicado en detalle todos los aspectos necesarios para hacer una aplicación J2ME utilizando códigos bidimensionales, vamos a explicar dos bloques fundamentales en la creación de la funcionalidad de QR-APDID. Son otros dos paquetes opcionales más que nos permiten darle toda la funcionalidad que permita que este MIDlet funcione al 100%.

2.2.6 Otros paquetes opcionales

Después de haber visto, en un apartado anterior, lo más importante de MMAPI (**JSR 135,** un paquete opcional), lo que nos permite disponer de multimedia en nuestro MIDlet, nos interesa ver otros dos paquetes opcionales que nos dan funcionalidad variada para la aplicación. Uno de estos paquetes es el **JSR 75, el Personal Information Management and File Connection API,** también llamado **PDA Optional Packages API (**uso de FileConnection en la aplicación**),** que nos permite el acceso al sistema de ficheros de nuestro móvil, tanto al sistema local como a memorias establecidas en el teléfono. Esto es muy importante para QR-APDID ya que le da al administrador la posibilidad de ofrecer información a los usuarios con archivos que los mismos hayan descargado o establecido, previamente, en su móvil. Es decir, que con este paquete opcional, utilizamos datos establecidos en el teléfono móvil para mostrarlos por pantalla, lo que implica que estos datos tengan que existir en el móvil y es una alternativa de muestra de la información. El otro paquete opcional es el **JSR 172** ó **J2ME Web Services API (**uso de HttpConnection en la aplicación**),** que amplía la funcionalidad de nuestro MIDlet añadiendo una plataforma de acceso a servicios web. Estas APIs permiten que nuestros dispositivos J2ME puedan ser clientes de servicios web mediante un modelo de programación consistente con la plataforma estándar de servicios web.

Estas dos "opciones" son soportadas por QR-APDID y su uso es "opcional" por parte del administrador, es decir, que el administrador puede elegir entre estas opciones, o las dos o ninguna, según la información que quiera prestar a sus usuarios/clientes. Evidentemente si el administrador no sirve información multimedia (audio ó vídeo) no hará falta ninguna de estas dos opciones (no es obligatorio aunque podría utilizarlas). La cosa cambia si se sirve información multimedia, entonces es aquí donde el administrador debe elegir entre la opción que mejor le convenga. Dentro de este contexto el cliente tiene que saber, de antemano, como va a recibir esta información para poder interactuar con el entorno dentro del contexto especificado.
Aquí se abre una doble contextualización del entorno: el primero, un entorno en el que el administrador prefiere no disponer de servidores y deja que el cliente se descargue de una manera concreta (ya sea vía web en su casa, o de cualquier otra forma) la información que, una vez dentro del entorno determinado, el usuario visualizará según el CQR que esté leyendo. Por otro lado, este contexto puede ser opcional por parte del cliente, en un entorno en el que el administrador da la posibilidad de utilizar las dos formas (o por archivos o vía web) y el usuario, por cualquier motivo, prefiere leer los archivos guardados, previamente, en su terminal (un motivo claro para realizar esta acción sería en el caso de no disponer de un teléfono sin conexión wifi, ya que utilizar 3G implicaría tener que pagar costes adicionales para utilizar QR-APDID). El segundo, un entorno en el que existe un servidor donde el administrador almacena la información que quiere servir a los usuarios/clientes. Sencillamente el usuario lee los CQR y la información se encuentra en el servidor correspondiente (alternativa al otro caso o complemento del mismo). De todos modos, cabe destacar que si la información no es multimedia, QR-APDID permite que la información se extraiga directamente del CQR (imagen codificada en Base64 ó texto).

En definitiva, estos dos paquetes opcionales dan una funcionalidad a QR-APDID cuyo uso es opcional por parte del administrador y los usuarios/clientes (según el contexto en el que se muevan todos ellos), lo que da a esta aplicación una dimensión aun mayor de la que tenía con sólo el soporte de información multimedia que, por otra parte, sí que es obligatorio su uso, aunque sea opcional el MMAPI.
Por último, apuntar que hay una serie de paquetes que se deben importar para poder dar a nuestro MIDlet la posibilidad de adquirir dichos comportamientos.

2.3 Servlet

El servlet es una tecnología que corre dentro del contexto de un cotenedor web como es, por ejemplo, Tomcat (más adelante explicaremos su funcionalidad y detalles) o un servidor, y extienden la funcionalidad de los mismos. Su uso más común es el de generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web (en nuestro caso, no es exactamente este el funcionamiento pero es muy parecido, ya que coge los parámetros de la llamada para buscar el archivo y enviarlo al cliente, el terminal móvil). Un servlet es un objeto que se ejecuta en un servidor o contenedor J2EE, especialmente diseñado para ofrecer contenido dinámico desde un servidor web, generalmente HTML. Esta es una alternativa a otras opciones que permiten generar contenido dinámico, como son los lenguajes ASP, PHP, JSP (un caso especial de servlet), Ruby y Python. Forman parte de J2EE (Java Enterprise Edition), que es una ampliación de J2SE (Java Standard Edition) para la creación de entornos empresariales.

Un servlet implementa la interfaz "javax.servlet.Servlet" o hereda alguna de las clases más convenientes para un protocolo específico (ejemplo: javax.servlet.HttpServlet). Al implementar esta interfaz, el servlet es capaz de interpretar los objetos de tipo HttpServletRequest y HttpServletResponse, los cuáles contienen la información de la página que invocó al *servlet*. Entre el servidor de aplicaciones (o "web content") y el *servlet* existe un contrato que determina cómo han de interactuar. La especificación de éste se encuentra en los JSR (Java Specification Requests) que hemos visto previamente ([1] y [2]).

El ciclo de vida de un Servlet se divide en los siguientes puntos ([3]):

1. El cliente solicita una petición a un servidor vía URL.
2. El servidor recibe la petición.
    1. Si es la primera, se utiliza el motor de Servlets para cargarlo y se llama al método `init()`.
    2. Si ya está iniciado, cualquier petición se convierte en un nuevo hilo. Un Servlet puede manejar múltiples peticiones de clientes.
3. Se llama al método `service()` para procesar la petición devolviendo el resultado al cliente.
4. Cuando se apaga el motor de un Servlet se llama al método `destroy()`, que lo destruye y libera los recursos abiertos.

Este pequeño resumen viene explicado de forma detallada a continuación:

La clase GenericServlet es una clase abstract puesto que su método service() es abstract. Esta clase implementa dos interfaces, de las cuales la más importante es la interface Servlet.

La interface Servlet declara los métodos más importantes de cara a la vida de un servlet: init() que se ejecuta sólo al arrancar el servlet; destroy() que se ejecuta cuando va a ser destruido y service() que se ejecutará cada vez que el servlet deba atender una solicitud de servicio. Cualquier clase que derive de GenericServlet deberá definir el método service(). Es muy interesante observar los dos argumentos que recibe este método, correspondientes a las interfaces ServletRequest y ServletResponse. La primera de ellas referencia a un objeto que describe por completo la solicitud de servicio que se le envía al servlet. Si la solicitud de servicio viene de un formulario HTML, por medio de ese objeto se puede acceder a los nombres de los campos y a los valores introducidos por el usuario; puede también obtenerse cierta información sobre el cliente (ordenador y browser). El segundo argumento es un objeto con una referencia de la interface ServletResponse, que constituye el camino mediante el cual el método service() se conecta de nuevo con el cliente y le comunica el resultado de su solicitud. Además, dicho método deberá realizar cuantas operaciones sean necesarias para desempeñar su cometido: escribir y/o leer datos de un fichero, comunicarse con una base de datos, etc. El método service() es realmente el corazón del
servlet.

En la práctica, salvo para desarrollos muy especializados, todos los servlets deberán construirse a partir de la clase HttpServlet, sub-clase de GenericServlet.

La clase HttpServlet ya no es abstract y dispone de una implementación o definición del método service(). Dicha implementación detecta el tipo de servicio o método HTTP que le ha sido solicitado desde el browser y llama al método adecuado de esa misma clase (doPost(), doGet(), etc.). Cuando el programador crea una sub-clase de HttpServlet, por lo general no tiene que redefinir el método service(), sino uno de los métodos más especializados (normalmente doPost()), que tienen los mismos argumentos que service(): dos objetos referenciados por las interfaces ServletRequest y ServletResponse.

También hay otras interfaces utilizables en java que son las resumidas a continuación:

1. La interface ServletContext permite a los servlets acceder a información sobre el entorno en que se están ejecutando.

2. La interface ServletConfig define métodos que permiten pasar al servlet información sobre sus parámetros de inicialización.

3. La interface ServletRequest permite al método service() de GenericServlet obtener información sobre una petición de servicio recibida de un cliente. Algunos de los datos proporcionados por GenericServlet son los nombres y valores de los parámetros enviados por el formulario HTML y una input stream.

4. La interface ServletResponse permite al método service() de GenericServlet enviar su respuesta al cliente que ha solicitado el servicio. Esta interface dispone de métodos para obtener un output stream o un writer con los que enviar al cliente datos binarios o caracteres, respectivamente.

5. La interface HttpServletRequest deriva de ServletRequest. Esta interface permite a los métodos service(), doPost(), doGet(), etc. de la clase HttpServlet recibir una petición de servicio HTTP. Esta interface permite obtener información del header de la petición de servicio HTTP.

6. La interface HttpServletResponse extiende ServletResponse. A través de esta interface los métodos de HttpServlet envían información a los clientes que les han pedido algún servicio.

El API del JSDK dispone de clases e interfaces adicionales, no citadas en esta memoria.

A continuación vemos las características más importantes de los servlets:

1. Son independientes del servidor utilizado y de su sistema operativo, lo que quiere decir que a pesar de estar escritos en Java, el servidor puede estar escrito en cualquier lenguaje de programación, obteniéndose exactamente el mismo resultado que si lo estuviera en Java.

2. Los servlets pueden llamar a otros servlets, e incluso a métodos concretos de otros servlets. De esta forma se puede distribuir de forma más eficiente el trabajo a realizar. Por ejemplo, se podría tener un servlet encargado de la interacción con los clientes y que llamara a otro servlet para que a su vez se encargara de la comunicación con una base de datos. De igual forma, los servlets permiten redireccionar peticiones de servicios a otros servlets (en la misma máquina o en una máquina remota).

3. Los servlets pueden obtener fácilmente información acerca del cliente (la permitida por el protocolo HTTP), tal como su dirección IP, el puerto que se utiliza en la llamada, el método utilizado (GET, POST, etc), etc.

4. Permiten además la utilización de cookies y sesiones, de forma que se puede guardar información específica acerca de un usuario determinado, personalizando de esta forma la interacción cliente-servidor. Una clara aplicación es mantener la sesión con un cliente.

5. Los servlets pueden actuar como enlace entre el cliente y una o varias bases de datos en arquitecturas cliente-servidor de 3 capas (si la base de datos está en un servidor distinto).

6. Asimismo, pueden realizar tareas de proxy para un applet. Debido a las restricciones de seguridad, un applet no puede acceder directamente por ejemplo a un servidor de datos localizado en cualquier máquina remota, pero el servlet sí puede hacerlo de su parte.

7. Al igual que los programas CGI, los servlets permiten la generación dinámica de código HTML dentro de una propia página HTML. Así, pueden emplearse servlets para la creación de contadores, banners, etc.

El JSDK (Java Servlet Developer Kit), distribuido gratuitamente por Sun, proporciona el conjunto de herramientas necesarias para el desarrollo de servlets. El JSDK 2.0 se encuentra disponible en la dirección de Internet http://java.sun.com/products/servlet/index.html.

El JSDK consta básicamente de 2 partes:

1. El API del JSDK, que se encuentra diseñada como una extensión del JDK propiamente dicho. Consta de dos packages que se encuentran contenidos en javax.servlet y javax.servlet.http. Este último es una particularización del primero para el caso del protocolo http (utilizado en la aplicación QRMidlet), al ser el más extendido en la actualidad. Mediante este diseño lo que se consigue es que se mantenga una puerta abierta a la utilización de otros protocolos que existen en la actualidad (FTP, POP, SMTP, etc.), o vayan siendo utilizados en el futuro. Estos packages están almacenados en un fichero JAR (\lib\jsdk.jar).

2. La documentación propiamente dicha del API y el código fuente de las clases (similar a la de los JDK 1.1 y 1.2).

Podemos crear un Servlet haciendo uso del paquete “javax.servlet.”. Para la creación y compilación del mismo y para su posterior funcionamiento han hecho falta 2 importantes entornos de trabajo: para su desarrollo, Eclipse, y para su despliegue y puesta en funcionamiento, Apache-Tomcat. Estas dos tecnologías serán vistas en detalle en los próximos apartados. Se les da bastante importancia ya que sin ellos sería imposible dotar de la posibilidad de accesibilidad web a la aplicación QR-APDID, una accesibilidad opcional que permite, tanto a clientes/usuarios como administradores, utilizar este lector de CQR para conseguir extraer la información que los mismos codifican y que ésta se encuentra alojada en un servidor (como veremos, Tomcat nos ayuda con este apartado). Además veremos un poco de la historia de cada una de estas tecnologías (por cultura general) y los posibles usos que tienen a parte del que nos compete. Pero nos centraremos, básicamente, en lo que nos interesa de cada tecnología en referencia a la utilización de servlets y la forma en que se trabajan con ellos.

2.4 Eclipse

Para el desarrollo concreto del servlet, Eclipse es la herramienta más interesante. Eclipse es una potente y completa plataforma de programación, desarrollo y compilación de elementos tan variados como sitios web, programas en C++ o aplicaciones Java.

Eclipse es un entorno de desarrollo integrado (IDE) en el que se pueden encontrar todas las herramientas y funciones necesarias para el trabajo, recogidas además en una atractiva interfaz que lo hace fácil y agradable de usar.

Eclipse cuenta con un editor de texto donde se puede ver el contenido del fichero en el que se está trabajando, una lista de tareas, y otros módulos similares.

Si bien las funciones de Eclipse son más bien de carácter general, las características del programa se pueden ampliar y mejorar mediante el uso de plug-ins.

Hay otras herramientas semejantes que gozan de gran popularidad en este ámbito, como es el caso de NetBeans.

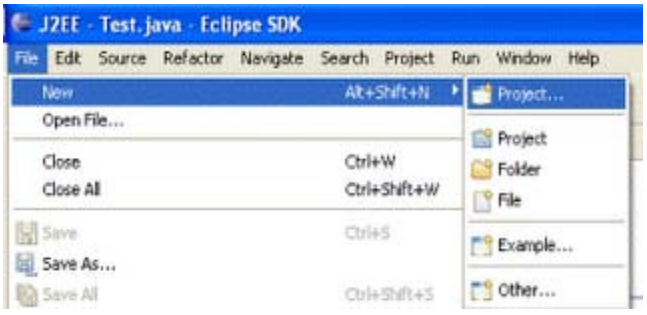
2.4.1 Desarrollo de Servlets con Eclipse

A continuación veremos cómo crear un Servlet en un proyecto Web haciendo uso del IDE Eclipse. Primeros requisitos antes de empezar:

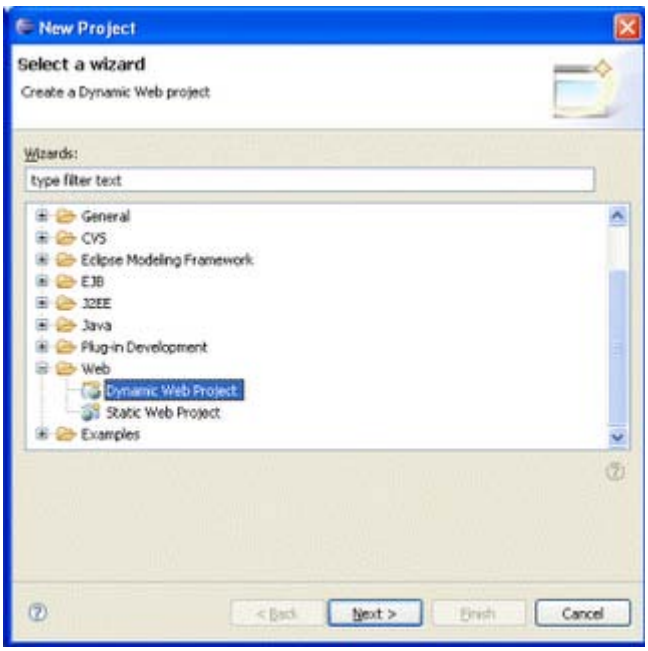
- Instalar JDK.
- Instalar un Servidor Web de aplicaciones para Java como, por ejemplo, Apache Tomcat.
- Instalación de Eclipse.

Con todo esto hecho, ahora vamos a proceder con la creación de un proyecto web con Eclipse:

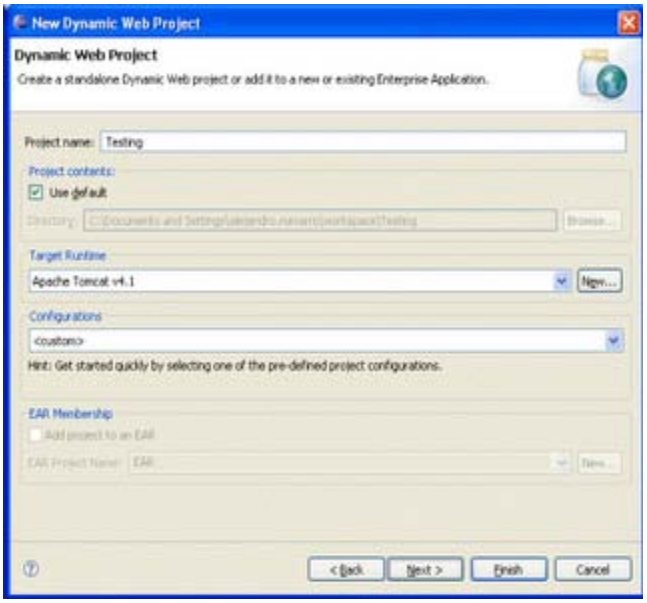
Clic en el menú principal. **File - New - Project...**



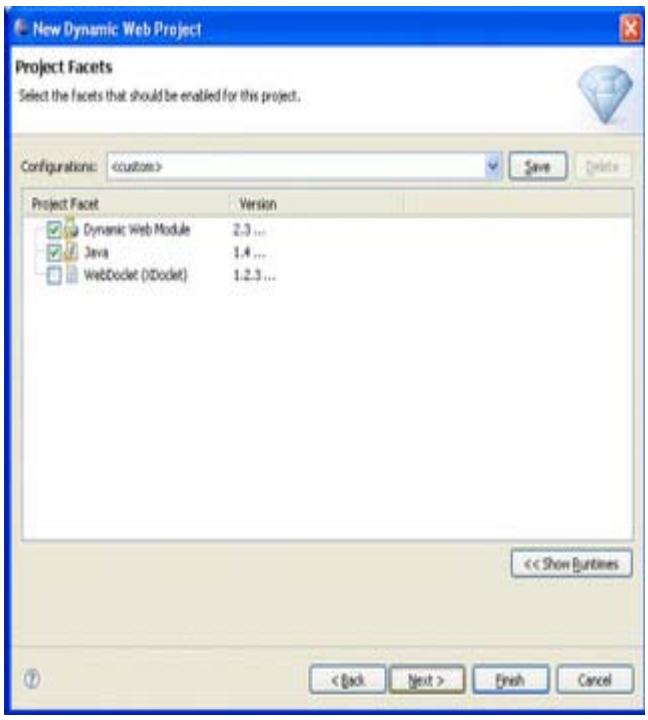
Ahora nos aparece una nueva ventana, damos clic en la carpeta **Web** y seleccionamos la opción **Dynamic Web Project**, ahora oprimimos el botón **Next**.



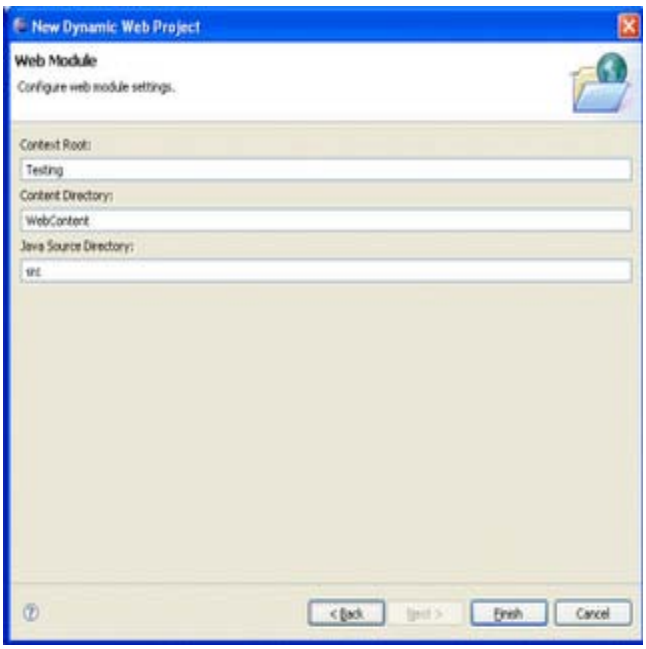
En esta nueva ventana establecemos el nombre de nuestro proyecto y seleccionamos el Servidor Web. Para nuestro ejemplo el nombre del proyecto es **Testing** y vamos a hacer uso del servidor Apache Tomcat 4.1. Damos clic en el botón **Next**.



En esta otra ventana se nos indican las versiones que se utilizaron como por ejemplo, se nos indica que se hará uso de la versión 1.4 de Java. Clic en el botón **Next**.



A continuación, se visualiza la pantalla de configuración final para nuestra aplicación Web. Aquí se nos indica el directorio donde se almacenarán nuestros archivos, así como el Contexto con el que haremos referencia a nuestra aplicación, para nuestro ejemplo el Contexto es **Testing** y con ello, para hacer referencia a nuestra aplicación en el servidor local, lo podremos hacer de la siguiente manera: http://localhost/Testing/. Damos clic en el botón **Finish**.

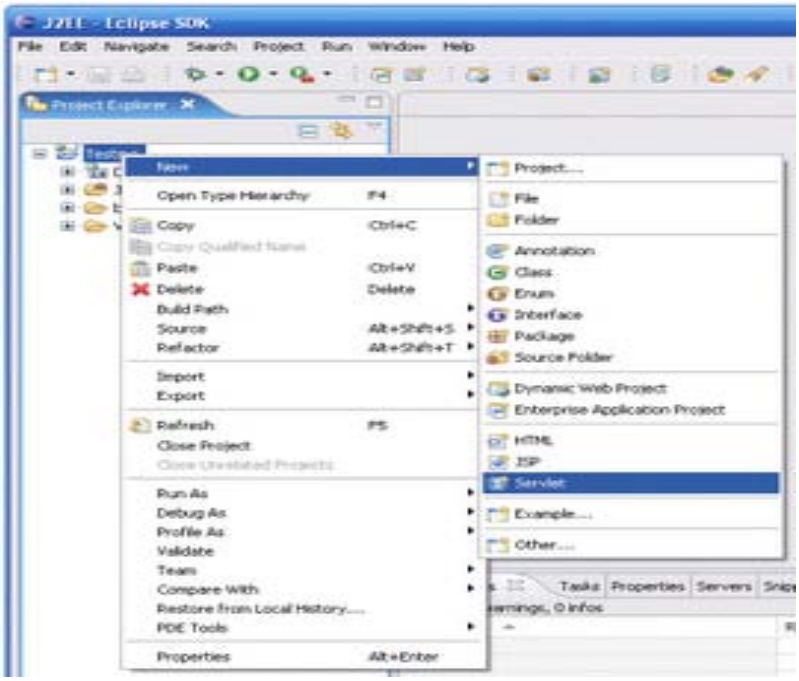


Con esto ya hemos creado nuestro proyecto Web.

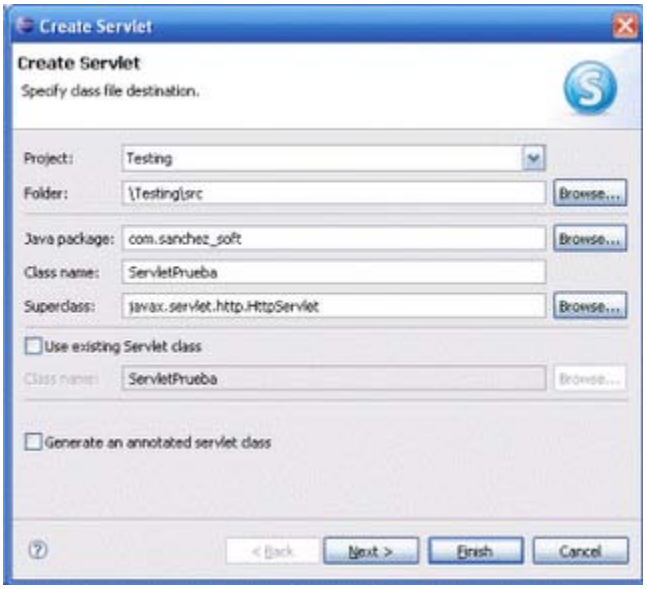
NOTA: este último paso hay que hacerlo pero para nuestro desarrollo no es relevante (se puede hacer todo con Eclipse, pero también se puede optar por no hacerlo así. Esto es una opinión puramente personal). Para hacer las pruebas referentes en el

servidor, también se pueden hacer directamente con Apache-Tomcat. De hecho, más adelante explicaremos cómo se hace, y es de esa forma como lo he hecho yo. Estos pasos son sencillamente para poder desarrollar el servlet en un entorno de trabajo adecuado como es Eclipse. Una vez terminado su desarrollo, simplemente hay que coger los archivos necesarios, meterlos en los directorios oportunos de Apache-Tomcat y empezar desde ahí su despliegue y dar una verdadera referencia a la aplicación web.

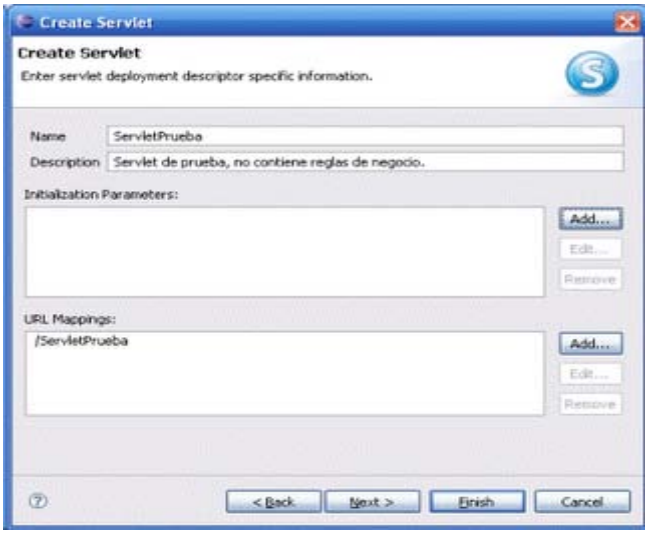
Una vez que contamos con lo anterior, procedemos a abrir nuestro proyecto Web y damos clic con el botón secundario del ratón sobre él, seleccionamos la opción **New** y, posteriormente, click en la opción SERVLET**:**



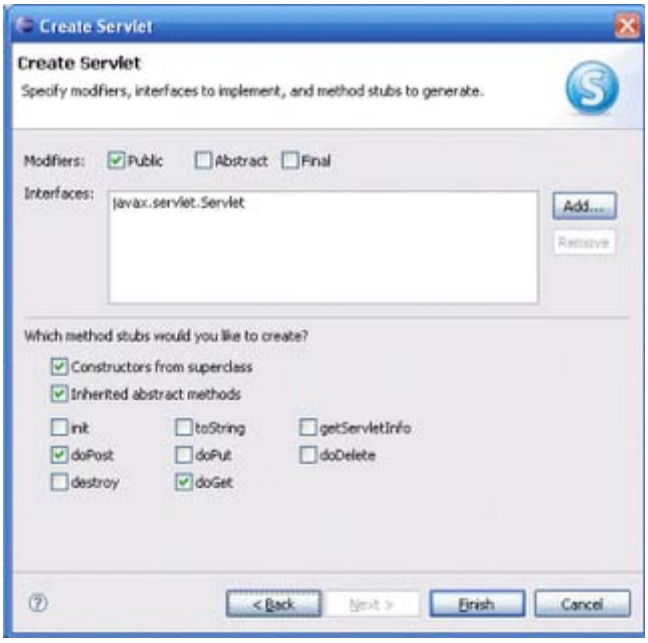
Ahora nos aparecerá la siguiente ventana, en la cual se nos solicita el nombre del Servlet que deseamos crear, así como el paquete en el cual estará contenido. Para nuestro ejemplo, el nombre del servlet es ServletPrueba (porque estamos en un ejemplo genérico) y se encontrará en el paquete sanchez_soft. A continuación damos clic en el botón Next.



A continuación nos aparecerá la siguiente pantalla, en la cual indicaremos con qué nombre o alias queremos hacer referencia a este servlet desde nuestro sitio Web, generalmente se deja el mismo nombre de la clase Servlet que creamos. Damos clic en el botón **Next**.

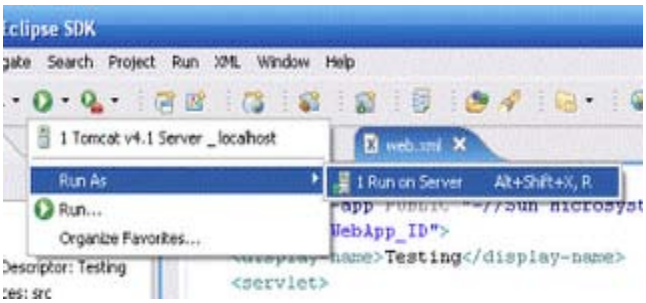


En la siguiente pantalla se nos brinda la opción para que Eclipse cree de manera automática algunos métodos. Damos clic en el botón **Finish**.



Eclipse creerá el esqueleto de nuestro Servlet, ahora nos corresponde agregar el código del servlet para dotar de funcionalidad al mismo cuando lo ejecutemos.

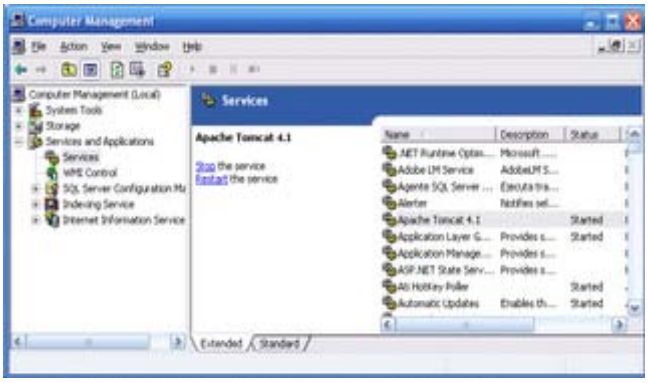
Una vez que hemos escrito el código, para visualizar el resultado tenemos dos posibilidades: la primera es ejecutar el Servlet como se muestra en la siguiente imagen y seguir con los siguientes pasos donde se utiliza todo el rato Eclipse y la segunda, compilar y ejecutar el Servlet, coger el fichero ".class" y, como apuntamos antes, establecer un contexto en Apache-Tomcat, organizar los ficheros y hacer un despliegue (esto viene explicado más adelante y, como dijimos anteriormente, esta ha sido la opción elegida para la creación final del servidor).



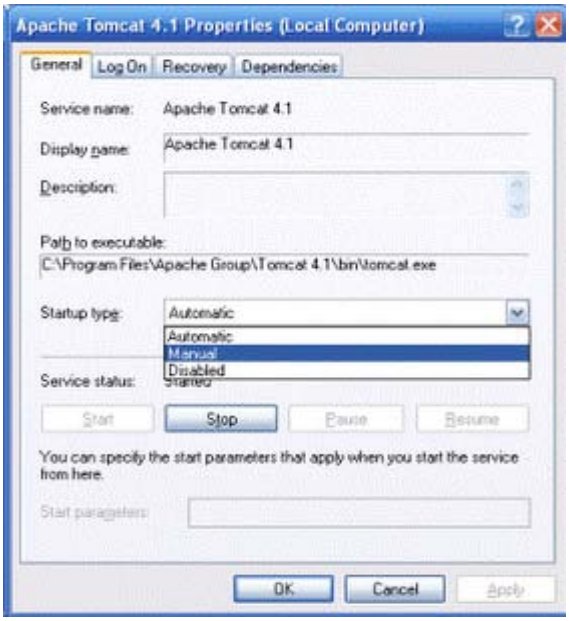
Atención: un error sumamente común es tener el servidor de aplicaciones corriendo e intentar ejecutar nuestra aplicación Web con Eclipse, si realizamos esto nos aparecerá un mensaje de error indicándonos que el servicio ya esta iniciado y que no se puede hacer otra petición en los mismos puertos

Para solucionar esto y evitar que esto nos ocurra constantemente, debemos apagar nuestro servidor y configurar para que el encendido sea manual y no automático. Para ello seguir las siguientes instrucciones: Acceder a la Administración de Servicios.

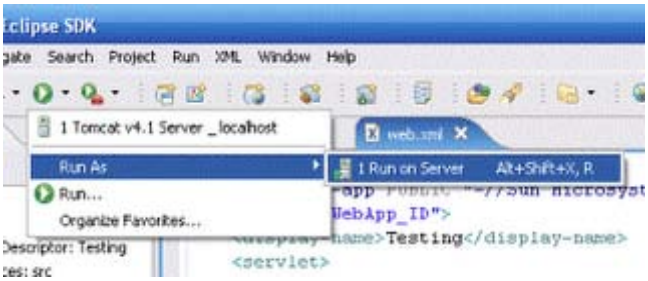
Seleccionar el servicio de Apache Tomcat (o el del server que estén ejecutando). Damos clic en él con el botón secundario del ratón y clic en la opción de Propiedades.



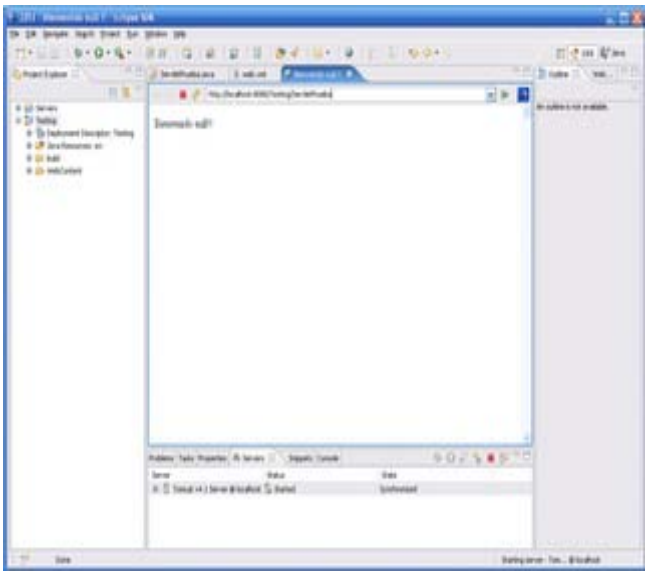
En la ventana de propiedades seleccionamos el tipo de inicio como **Manual** y paramos el servicio dando clic en el botón **Stop**, y finalmente damos clic en el botón **Ok**.



Ahora si regresamos a nuestra aplicación y la ejecutamos.



Una vez ejecutado nuestro servlet, nos aparecerá una pantalla similar a la siguiente:



Ya hemos creado un Servlet y realizado la lectura del valor de un parámetro que recibimos por URL. Ahora vamos a proceder a explicar el contenedor Apache-Tomcat y, de forma detallada, vamos a estudiar el desarrollo del servlet en esta tecnología que, como hemos dicho antes, a partir de un cierto punto se puede hacer todo directamente

en Tomcat. De todos modos está bien tener dos posibilidades distintas de poder desarrollar un servlet, dando al desarrollador la oportunidad de elegir. A priori, parece más sencillo e interesante utilizar Eclipse para todo y no solo para conseguir el fichero ".class" del servlet que es, en definitiva, lo único para lo que utilizamos este programa en la construcción del mismo para el desarrollo final de QR-APDID (con el fichero ".class" ya podemos ir a Tomcat y desplegar toda nuestra infraestructura web, además de todos los ficheros que se quieran conservar en el servidor). Por esto, damos las pautas específicas para poder hacer todo el desarrollo con Eclipse. Como al final esta no fue la opción escogida, en los apartados siguientes explicamos todo lo referente a Apache-Tomcat y el desarrollo de los servlets en él (**[7]**).

2.5 Apache-Tomcat

Apache Tomcat ([13]) funciona como un contenedor de servlets que es mantenido y desarrollado por miembros de la Apache Software Foundation y voluntarios independientes. Los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la *Apache Software Licence*. Estamos ante un servidor web con soporte de servlets y JSPs que no es un servidor de aplicaciones (como puede ser, por ejemplo, JBoss). Entre sus características más importantes están la de que el motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache y, además, puede funcionar como servidor web por sí mismo. En sus inicios existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. Como curiosidad, saber que Tomcat fue escrito en Java, por lo que funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

A continuación se presentan la estructura de directorios de Apache-Tomcat, importantes para la correcta construcción de un entorno web determinado. A parte, se da un esbozo de las claves necesarias para la creación de la parte web de QR-APDID, donde se utiliza Tomcat y el servlet que nos sirve cierto tipo de información almacenada en el servidor. Para la versión de Windows XP (la versión utilizada para realizar el desarrollo completo del proyecto) se deben seguir las siguientes pautas de instalación (sacadas de un post de "http://jcalderon.wordpress.com"):

Se debe descargar la versión reciente de Java Platform, Standard Edition 6 Development Kit (JDK 6). Debe instalarse previo a la instalación de Apache-Tomcat. Apache-tomcat-6.0.14. No se debe descargar la versión que lo instala como un servicio de Windows si se va a usar para desarrollar. Lo recomendable es descargar la versión que se instala manualmente descomprimiendo un archivo de tipo zip (Windows) o un tar.gz (Linux) en la página oficial de tomcat.

En nuestro caso cogeremos el archivo ".rar".

Se descomprime el archivo y se accede al directorio donde se ha descomprimido, donde quedará una jerarquía como la siguiente:

$CATALINA_HOME = Representa la raíz donde se ha instalado apache-tpmcat. Cada uno de los directorios de la imagen anterior posee su respectivo significado:

- bin - arranque, cierre, y otros scripts y ejecutables para la gestión de nuestro sitio web.
- common- clases comunes que pueden utilizar Catalina y las aplicaciones web
- conf – ficheros XML y los correspondientes DTD para la configuración de Tomcat
- logs - logs de Catalina y de las aplicaciones
- server - clases utilizadas solamente por Catalina
- shared - clases compartidas por todas las aplicaciones web
- webapps - directorio que contiene las aplicaciones web
- work - almacenamiento temporal de ficheros y directorios

El siguiente paso es uno de los más importantes de la instalación de Tomcat ya que debemos crear la variable de entorno JAVA_HOME. Ejemeplo:

JAVA_HOME = directorio del JDK.

Para acceder a la creación de la variable de entorno: Windows+Pausa>Opciones Avanzadas>Variables de Entorno>Nueva

- Existen dos archivos sumamente importantes los cuales nos permitirán iniciar y parar apache-tomcat se encuentra en el siguiente directorio.

$CATALINA_HOME/bin/startup = para iniciar o arrancar
$CATALINA_HOME/bin/shutdown = para parar o detener

- Seguidamente se abre un navegador web y para escribir una URL del siguiente tipo:

http://{host}:{port}/ = donde {host}{port} representa el hostname y el puerto donde corre apache-tomcat, entonces quedaría http://localhost:8080/ y aparecerá la pagina de bienvenida de apache-tomcat.

Nota: Se debe recordar que hay que iniciar apache-tomcat o de lo contrario se producirá un ERROR.

- Para poder acceder a las aplicaciones de gestión y administración de apache-tomcat es necesario crear un usuario accediendo al siguiente directorio.

$CATALINA_HOME/conf/tomcat-users.xml = este archive permite crear un username, password and roles.

Así es como se debería poder empezar a trabajar con el servidor Apache-Tomcat. Aún así, ante cualquier problema de instalación o funcionamiento, se recomienda leer los tutoriales oficiales y la página web oficial.

Hay un montón de versiones de Tomcat por lo que también hay evolución en este sistema de directorios (pocos cambios entre unas versiones y otras). Las últimas versiones son las referidas a Tomcat 7.x.

Ahora vamos a ver el desarrollo del servlet en Tomcat.

2.5.1 Desarrollo de un servlet en Tomcat

Primero procedemos a la instalación y configuración de Apache-Tomcat:

- Descargar el fichero de tomcat.
- Extraer los ficheros de Tomcat en algún directorio conocido.
- Establecer la variable de entorno *CATALINA_HOME* para que apunte al directorio donde hemos instalado nuestro servidor Tomcat.
- Añadir el directorio *${CATALINA_HOME}/bin* a la variable de entorno *PATH*.
- Establecer la variable de entorno *JAVA_HOME* para que apunte al directorio donde está instalado Java 1.4.2
- Añadir el directorio *${JAVA_HOME}/bin* a la variable de entorno *PATH*. Ahora ya podemos ejecutar el servidor Tomcat y empezar a probar el servlet.
- Instrucciones de arranque y parada de Tomcat: El script de arranque se encuentra en el directorio *bin* de la instalación del Tomcat, como hemos incluido esta ruta en nuestra variable de entorno *PATH* será suficiente ejecutar en la shell: startup.sh. Del mismo modo, para parar el servidor se debe ejecutar el siguiente comando en la shell: shutdown.sh. Para comprobar que la instalación ha sido satisfactoria, hay que arrancar el servidor Tomcat y desde un navegador conectaros a la URL: *http://localhost:8080*. Si se ve la página principal de Tomcat, la instalación ha sido correcta.

Una vez comprobado que la configuración es correcta, debemos tener en cuenta que cuando vayamos a usar la aplicación en el móvil, la url debe ser la de la máquina donde se aloje el servidor, nunca el “localhost”.

En este momento vendría todo lo referente al despliegue del servlet en Tomcat:

1.- Codificar y compilar el servlet: Una vez creado, para compilar el servlet, se necesita la API de Servlet (es en este punto donde utilicé Eclipse, para la creación, compilación y ejecución del servlet, así conseguía el fichero ".class", muy necesario para seguir con el desarrollo del servlet, pero Tomcat da también la posibilidad de hacerlo directamente sobre él. A partir de entonces es cuando utilicé Tomcat, simplemente para el despliegue, comentado en el punto siguiente), disponible en la instalación de Tomcat, en el fichero *common/lib/servlet.jar*. Los fabricantes recomiendan incluirla en la variable de entorno *CLASSPATH*, o compilar con la opción *-classpath*, por ejemplo:

javac -classpath ${CATALINA_HOME}/common/lib/servlet.jar:. MyServlet.java

2.- Desplegar el servlet en Tomcat: consiste en incluir una serie de ficheros en un contenedor web (por ejemplo, Tomcat) para que los clientes puedan acceder a su funcionalidad. Normalmente, el desarrollo de un servlet forma parte de lo que se denomina una aplicación Web, que no es más que una colección de servlets, páginas HTML, JSP, clases y otros recursos que se pueden empaquetar y ejecutar en distintos contenedores web, de distintos fabricantes, y que ofrecen una determinada funcionalidad a la que los clientes acceden típicamente a través de un navegador. Las Aplicaciones Web, deben estructurarse según la siguiente jerarquía de subdirectorios: *Directorio raíz:* puedes publicar ficheros estáticos (HTML, imágenes, hojas de estilo, etc.) y JSPs.

- Directorio *WEB-INF:* debe contener un fichero *web.xml*. Este fichero configura la aplicación. Por ejemplo, permite declarar servlets, asignarles parámetros de inicio, declarar alias y filtros, etc. En nuestro caso, solo se declara el servlet con su nombre y la url local del mismo en el servidor, con el que se puede acceder a él desde el exterior.
  - Directorio *classes*: Aquí se colocan los ficheros compilados (servlets, beans, etc.) de las clases utilizadas por la aplicación web (MyServlet.java).
  - Directorio *lib*: puedes colocar en él otras bibliotecas de clases adicionales (comprimidas con jar) que utilice la aplicación (en nuestro caso no será necesario).
- Resto de subdirectorios: para ficheros estáticos y JSP (que en este caso no serán necesarios, pues sólo utilizaremos un servlet).

Para desplegar nuestro servlet creamos un contexto para nuestra aplicación web, denominado MYSERVER (crear un contexto en Tomcat no es más que crear un directorio debajo de *${CATALINA_HOME}/webapps/).* Este directorio se corresponde con el directorio raíz de la aplicación web y, por lo tanto, debajo de él creamos la estructura de subdirectorios indicada anteriormente. De hecho, el nombre del contexto es el primer nivel de la jerarquía de la ruta de acceso a nuestra aplicación:

http://localhost:8080/MYSERVER/lugar_del_recurso

3.- Ejecutar el servlet: Si arrancamos un navegador y nos conectamos a la URL *http://localhost:8080/MYSERVER/serv/* activaremos nuestro servlet y se mostrarán los errores de conexión. Eso nos indicará que nos estamos conectando, correcta o incorrectamente (según se mire), al servlet. Esta no es la manera de acceder al mismo ya que la buena manera es mandando un parámetro y el servlet devuelve (o no) unos datos que sirven para crear una respuesta en el cliuente de la petición, que es el móvil (en este caso). Pero es importante constatar que la conexión se puede realizar correctamente ([7]).

2.6 MP-3

MPEG-1 Audio Layer III o ***MPEG-2 Audio Layer III***, más comúnmente conocido como **MP3**, es un formato de compresión de audio digital patentado que usa un algoritmo con pérdida para conseguir un menor tamaño de archivo. Es un formato de audio común usado para música tanto en ordenadores como en reproductores de audio portátil ([1]).

MP3 fue desarrollado por el Moving Picture Experts Group (MPEG) para formar parte del estándar MPEG-1 y del posterior y más extendido MPEG-2. Un MP3 creado usando una compresión de 128kbit/s tendrá un tamaño de aproximadamente unas 11 veces menor que su homónimo en CD. Un MP3 también puede comprimirse usando una mayor o menor tasa de bits por segundo, resultando directamente en su mayor o menor calidad de audio final, así como en el tamaño del archivo resultante.

Este formato fue desarrollado principalmente por Karlheinz Brandenburg, director de tecnologías de medios electrónicos del Instituto Fraunhofer IIS, perteneciente al Fraunhofer-Gesellschaft - red de centros de investigación alemanes - que junto con Thomson Multimedia controla el grueso de las patentes relacionadas con el MP3. La primera de ellas fue registrada en 1986 y varias más en 1991. Pero no fue hasta julio de 1995 cuando Brandenburg usó por primera vez la extensión .mp3 para los archivos relacionados con el MP3 que guardaba en su ordenador. Un año después su instituto ingresaba en concepto de patentes 1,2 millones de euros. Diez años más tarde esta cantidad ha alcanzado los 26,1 millones.

Tras el desarrollo de reproductores autónomos, portátiles o integrados en cadenas musicales (estéreos), el formato MP3 llega más allá del mundo de la informática.

El formato MP3 se convirtió en el estándar utilizado para *streaming* de audio y compresión de audio con pérdida de mediana fidelidad gracias a la posibilidad de

ajustar la calidad de la compresión, proporcional al tamaño por segundo (bitrate), y por tanto el tamaño final del archivo, que podía llegar a ocupar 12 e incluso 15 veces menos que el archivo original sin comprimir.

Fue el primer formato de compresión de audio popularizado gracias a Internet, ya que hizo posible el intercambio de ficheros musicales. Los procesos judiciales contra empresas como Napster y AudioGalaxy son resultado de la facilidad con que se comparten este tipo de ficheros. A principios de 2002 otros formatos de audio comprimido como Windows Media Audio y Ogg Vorbis empiezan a ser masivamente incluidos en programas, sistemas operativos y reproductores autónomos, lo que hizo prever que el MP3 fuera paulatinamente cayendo en desuso, en favor de otros formatos, como los mencionados, de mucha mejor calidad. Uno de los factores que influye en el declive del MP3 es que tiene patente. Técnicamente, el tener una patente no significa que su calidad sea inferior ni superior, pero impide que la comunidad pueda seguir mejorándolo y puede obligar a pagar por la utilización de algún códec. Esto es lo que ocurre con los reproductores de MP3. Aun así, a finales de 2009, el formato mp3 continúa siendo el más usado y el que goza de más éxito, sacando nuevas versiones.

MP-3 es el tipo de archivo escogido para utilizar con la aplicación, debido a su gran uso actual en el día a día por parte de mucha gente. Ahora vamos a ver los detalles técnicos de este tipo de archivos:

En esta capa existen varias diferencias respecto a los estándares MPEG-1 y MPEG-2, entre las que se encuentra el llamado banco de filtros híbrido que hace que su diseño tenga mayor complejidad. Esta mejora de la resolución frecuencial empeora la resolución temporal introduciendo problemas de pre-eco que son predichos y corregidos. Además, permite calidad de audio en tasas tan bajas como 64 kbps.

Banco de filtros

El banco de filtros utilizado en esta capa es el llamado banco de filtros híbrido polifase / MDCT. Se encarga de realizar el mapeado del dominio del tiempo al de la frecuencia tanto para el codificador como para los filtros de reconstrucción del decodificador. Las muestras de salida del banco están cuantificadas y proporcionan una resolución en frecuencia variable, 6x32 o 18x32 subbandas, ajustándose mucho mejor a las bandas críticas de las diferentes frecuencias. Usando 18 puntos, el número máximo de componentes frecuenciales es: 32 x 18 = 576. Dando lugar a una resolución frecuencial de: 24000/576 = 41,67 Hz (si fs = 48 kHz.). Si se usan 6 líneas de frecuencia la resolución frecuencial es menor, pero la temporal es mayor, y se aplica en aquellas zonas en las que se espera efectos de pre-eco (transiciones bruscas de silencio a altos niveles energéticos).

La Capa III tiene tres modos de bloque de funcionamiento: dos modos donde las 32 salidas del banco de filtros pueden pasar a través de las ventanas y las transformadas MDCT y un modo de bloque mixto donde las dos bandas de frecuencia más baja usan bloques largos y las 30 bandas superiores usan bloques cortos. Para el caso concreto del MPEG-1 Audio Layer 3 (que concretamente significa la tercera capa de audio para el estándar MPEG-1) especifica cuatro tipos de ventanas: (a) NORMAL, (b) transición de ventana larga a corta (START), (c) 3 ventanas cortas (SHORT)

El modelo psicoacústico

La compresión se basa en la reducción del margen dinámico irrelevante, es decir, en la incapacidad del sistema auditivo para detectar los errores de cuantificación en condiciones de enmascaramiento. Este estándar divide la señal en bandas de frecuencia que se aproximan a las bandas críticas, y luego cuantifica cada subbanda en función del umbral de detección del ruido dentro de esa banda. El modelo psicoacústico es una modificación del empleado en el esquema II, y utiliza un método denominado predicción polinómica. Analiza la señal de audio y calcula la cantidad de ruido que se puede introducir en función de la frecuencia, es decir, calcula la "cantidad de enmascaramiento" o umbral de enmascaramiento en función de la frecuencia.

El codificador usa esta información para decidir la mejor manera de gastar los bits disponibles. Este estándar provee dos modelos psicoacústicos de diferente complejidad: el modelo I es menos complejo que el modelo psicoacústico II y simplifica mucho los cálculos. Estudios demuestran que la distorsión generada es imperceptible para el oído experimentado en un ambiente óptimo desde los 192 kbps y en condiciones normales. Para el oído no experimentado, o común, con 128 kbps o hasta 96 kbps basta para que se oiga "bien" (a menos que se posea un equipo de audio de alta calidad donde se nota excesivamente la falta de graves y se destaca el sonido de "fritura" en los agudos). En personas que escuchan mucha música o que tienen experiencia en la parte auditiva, desde 192 o 256 kbps basta para oír bien. La música que circula por Internet, en su mayoría, está codificada entre 128 y 192 kbps.

Codificación y cuantificación

La solución que propone este estándar en cuanto a la repartición de bits o ruido, se hace en un ciclo de iteración que consiste de un ciclo interno y uno externo. Examina tanto las muestras de salida del banco de filtros como el SMR (signal-to-mask ratio) proporcionado por el modelo psicoacústico, y ajusta la asignación de bits o ruido, según el esquema utilizado, para satisfacer simultáneamente los requisitos de tasa de bits y de enmascaramiento. Dichos ciclos consisten en:

Ciclo interno

El ciclo interno realiza la cuantización no-uniforme de acuerdo con el sistema de punto flotante (cada valor espectral MDCT se eleva a la potencia 3/4). El ciclo escoge un determinado intervalo de cuantización y, a los datos cuantizados, se les aplica codificación de Huffman en el siguiente bloque. El ciclo termina cuando los valores cuantizados que han sido codificados con Huffman usan menor o igual número de bits que la máxima cantidad de bits permitida.

Ciclo externo

Ahora el ciclo externo se encarga de verificar si el factor de escala para cada subbanda tiene más distorsión de la permitida (ruido en la señal codificada), comparando cada banda del factor de escala con los datos previamente calculados en el análisis psicoacústico. El ciclo externo termina cuando una de las siguientes condiciones se cumple:

- Ninguna de las bandas del factor de escala tiene mucho ruido.
- Si la siguiente iteración amplifica una de las bandas más de lo permitido.
- Todas las bandas han sido amplificadas al menos una vez.

Empaquetado o formateador de bitstream

Este bloque toma las muestras cuantificadas del banco de filtros, junto a los datos de asignación de bits/ruido y almacena el audio codificado y algunos datos adicionales en las tramas. Cada trama contiene información de 1152 muestras de audio y consiste de un encabezado, de los datos de audio junto con el chequeo de errores mediante CRC y de los datos auxiliares (estos dos últimos opcionales).

Un fichero Mp3 se constituye de diferentes frames MP3 que a su vez se componen de una cabecera Mp3 y los datos MP3. Esta secuencia de datos es la denominada "stream elemental". Cada uno de los Frames son independientes, es decir, una persona puede cortar los frames de un fichero MP3 y después reproducirlos en cualquier reproductor MP3 del Mercado. La cabecera consta de una palabra de sincronismo que es utilizada para indicar el principio de un frame válido. A continuación siguen una serie de bits que indican que el fichero analizado es un fichero Standard MPEG y si usa o no la capa 3. Después de todo esto, los valores difieren dependiendo del tipo de archivo MP3. Los rangos de valores quedan definidos en la ISO/IEC 11172-3.

En matemáticas, la transformada de Fourier discreta, designada con frecuencia por la abreviatura DFT (del inglés *discrete Fourier transform*), y a la que en ocasiones se denomina transformada de Fourier finita, es una transformada de Fourier ampliamente empleada en tratamiento de señales y en campos afines para analizar las frecuencias presentes en una señal muestreada, resolver ecuaciones diferenciales parciales y realizar otras operaciones, como convoluciones. Es utilizada en el proceso de elaboración de un fichero MP3.

La transformada de Fourier discreta puede calcularse de modo muy eficiente mediante el algoritmo FFT.

2.7 MP-4

MPEG-4 ([2]) es un formato contenedor especificado como parte del estándar internacional MPEG-4 de ISO/IEC. Se utiliza para almacenar los formatos audiovisuales especificados por ISO/IEC y el grupo MPEG (*Moving Picture Experts Group*) al igual que otros formatos audiovisuales disponibles. Se utiliza típicamente para almacenar datos en archivos para ordenadores, para transmitir flujos audiovisuales y, probablemente, en muchas otras formas.

Normalmente se puede cambiar, de manera segura, la extensión de los archivos de audio ".mp4" a ".m4a" (Extensión de Apple) y viceversa pero no así a ".mp3" ya que para poder ser reproducidos en un reproductor de audio, éste necesariamente debe tener la capacidad para descodificar el formato que está contenido en el fichero ".mp4" que generalmente está codificado en MPEG-4 AAC e incompatible con la codificación y descodificación de MPEG-1 Layer 3 para ".mp3".

MPEG-4 es una serie de códecs y estándares internacionales de vídeo, audio y datos creado especialmente para la web. Está formado por una serie algoritmos de compresión que codifica datos, audio, y vídeo optimizando su calidad de almacenamiento, codificación y distribución en redes. Con las cámaras de hoy, se integra, captura y codifica en una sola acción, lo que optimiza la potencialidad del usuario para emitir.

".mp4" permite transmitir flujos sobre Internet. También permite transmitir combinaciones de flujos de audio, vídeo y texto coordinado de forma consolidada. El punto de partida para este formato fue el formato de archivo de QuickTime de Apple.

En la actualidad ".mp4" se ha visto enriquecido en formas muy variadas de manera que ya no se podría afirmar que son el mismo formato.

".mp4" se utiliza con frecuencia como alternativa a ".mp3" en el iPod y en iTunes. La calidad del codec AAC que se almacena en ".mp4" es mayor que la de MPEG-1 Audio Layer 3, pero su utilización no es actualmente tan amplia como la de ".mp3".

Es posible enviar prácticamente cualquier tipo de datos dentro de archivos ".mp4" por medio de los llamados "flujos privados", pero los formatos recomendados, por razones de compatibilidad son:

- Vídeo: MPEG-4, MPEG-2 y MPEG-1
- Audio: MPEG-4 AAC, MP3, MP2, MPEG-1 Part 3, MPEG-2 Part 3, CELP (habla), TwinVQ (tasas de bit muy bajas), SAOL (midi)
- Imágenes: JPEG, PNG
- Subtítulos: MPEG-4 Timed Text, o el formato de texto xmt/bt (significa que los subtítulos tienen que ser traducidos en xmt/bt)
- Systems: Permite animación, interactividad y menús al estilo DVD

Estas son algunas de las extensiones utilizadas en archivos que contienen datos en el formato *.mp4:

- .mp4: extensión oficial para audio, vídeo y contenidos avanzados (ver más abajo)
- .m4a: Sólo para archivos de audio; los archivos pueden ser renombrados como .mp4, si bien no todos los expertos recomiendan esto.
- .m4p: FairPlay archivos protegidos
- .m4v: sólo vídeo (algunas veces se utiliza para flujos mpeg-4 de vídeo no especificados en la definición del formato)
- .3gp .3g2: utilizados por la telefonía móvil 3G, también puede almacenar contenido no directamente especificados en la definición de .mp4 (H.263, AMR, TX3G)

Mp4 es el tipo de dato utilizado para audio/vídeo en **QRApdid.**

2.8 Base64 y Ascii

ASCII es un código de caracteres basado en el alfabeto latino, tal como se usa en inglés moderno y en otras lenguas occidentales. Para obtener un poco de cultura general acerca de este tipo de caracteres, fue creado en 1963 por el Comité Estadounidense de Estándares como una refundición o evolución de los conjuntos de códigos utilizados entonces en telegrafía. Más tarde, en 1967, se incluyeron las minúsculas, y se redefinieron algunos códigos de control para formar el código conocido como **US-ASCII**.

El código ASCII utiliza 7 bits para representar los caracteres, aunque inicialmente empleaba un bit adicional (bit de paridad) que se usaba para detectar errores en la transmisión.

ASCII fue publicado como estándar por primera vez en 1967 y fue actualizado por última vez en 1986. En la actualidad define códigos para 33 caracteres no imprimibles, de los cuales la mayoría son caracteres de control obsoletos que tienen efecto sobre cómo se procesa el texto, más otros 95 caracteres imprimibles que les siguen en la numeración (empezando por el carácter espacio).

Casi todos los sistemas informáticos actuales utilizan el código ASCII o una extensión compatible para representar textos y para el control de dispositivos que manejan texto como el teclado.

Codificación:

1- Se toman 3 bytes de la entrada.

2- Los 24 bits resultantes se separan en grupos de 6 bits.

3- Cada uno de esos grupos representa un número entre 0 y 63 que se usa como índice en la siguiente tabla para ser sustituido por un carácter: “ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/”.

4- Esos cuatro caracteres se vuelcan en la salida.

Se repite este proceso con cada bloque de 3 bytes hasta llegar al último bloque. Si este último fuese menos de 3 bytes, se procedería a rellenar con ceros hasta cumplir

la regla. Por último, si los últimos 4 caracteres obtenidos son C0, C1, C2, y C3, se añade:

Bloque último de 1 byte -> C0, C1, =, =

Bloque último de 2 bytes -> C0, C1, C2, =

Bloque último de 3 bytes -> C0, C1, C2, C3

Decodificación:

Este proceso es inverso al anterior. Cada 4 caracteres producirán 3 bytes, con la precaución de interpretar los últimos 4 como queda dicho arriba.

Ejemplo detallado:

Vamos a codificar 0xef 0xba 0xbc 0x95 0x5f

Primero se agrupan de tres en tres rellenando el último bloque con ceros:

0xefbabc y 0x955f00.

Ahora escribimos cada bloque en binario: 111011111011101010111100 y 100101010101111100000000. Estos números se dividen en bloques de 6 bits y se halla su equivalencia decimal: 59 59 42 60 y 37 21 60 0. Los caracteres resultantes son: 7 7 q 8 y l V 8 A.

Como el último bloque es de 2 bytes la cadena resultante será: 77q8lV8=.

2.9 JavaServer Pages

JavaServer Pages (**JSP**) es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo. Esta tecnología es un desarrollo de la compañía Sun Microsystems. La Especificación JSP 1.2 fue la primera que se liberó y en la actualidad estádisponible la Especificación JSP 2.1 ([1]).

Las JSP's permiten la utilización de código Java mediante scripts. Además, es posible utilizar algunas acciones JSP predefinidas mediante etiquetas. Estas etiquetas

pueden ser enriquecidas mediante la utilización de Bibliotecas de Etiquetas (TagLibs o Tag Libraries) externas e incluso personalizadas.

JSP puede considerarse como una manera alternativa, y simplificada, de construirservlets. Es por ello que una página JSP puede hacer todo lo que un servlet puede hacer, y viceversa. Cada versión de la especificación de JSP está fuertemente vinculada a una versión en particular de la especificación de servlets.

El funcionamiento general de la tecnología JSP es que el Servidor de Aplicaciones interpreta el código contenido en la página JSP para construir el código Java del servlet a generar. Este servlet será el que genere el documento (típicamente HTML) que se presentará en la pantalla del Navegador del usuario.

El rendimiento de una página JSP es el mismo que tendría el servlet equivalente, ya que el código es compilado como cualquier otra clase Java. A su vez, la máquina virtual compilará dinámicamente a código de máquina las partes de la aplicación que lo requieran. Esto hace que JSP tenga un buen desempeño y sea más eficiente que otras tecnologías web que ejecutan el código de una manera puramente interpretada.

La principal ventaja de JSP frente a otros lenguajes es que el lenguaje Java es un lenguaje de propósito general que excede el mundo web y que es apto para crear clases que manejen lógica de negocio y acceso a datos de una manera prolija. Esto permite separar en niveles las aplicaciones web, dejando la parte encargada de generar el documento HTML en el archivo JSP.

Otra ventaja es que JSP hereda la portabilidad de Java, y es posible ejecutar las aplicaciones en múltiples plataformas sin cambios. Es común incluso que los desarrolladores trabajen en una plataforma y que la aplicación termine siendo ejecutada en otra.

Los servlets y Java Server Pages (JSPs) son dos métodos de creación de páginas web dinámicas en servidor usando el lenguaje Java. En ese sentido son similares a otros métodos o lenguajes tales como el PHP, ASP o los CGIs, programas que generan páginas web en el servidor. Sin embargo, se diferencian de ellos en otras cosas.

Para empezar, los JSPs y servlets se ejecutan en una máquina virtual Java, lo cual permite que, en principio, se puedan usar en cualquier tipo de ordenador, siempre que exista una máquina virtual Java para él. Cada servlet (o JSP, a partir de ahora lo usaremos de forma indistinta) se ejecuta en su propia hebra, es decir, en su propio contexto; pero no se comienza a ejecutar cada vez que recibe una petición, sino que persiste de una petición a la siguiente, de forma que no se pierde tiempo en invocarlo (cargar programa + intérprete). Su persistencia le permite también hacer una serie de cosas de forma más eficiente: conexión a bases de datos y manejo de sesiones, por ejemplo.

Las JSPs son en realidad una forma alternativa de crear servlets ya que el código JSP se traduce a código de servlet Java la primera vez que se le invoca y en adelante es el código del nuevo servlet el que se ejecuta produciendo como salida el código HTML que compone la página web de respuesta.

2.10 Otros elementos utilizados

Para el desarrollo de QR-APDID, también se utilizó TextPad, un sencillo editor de texto para la codificación de todas las demás clases java que no era el servlet del cual hemos hablado ya. Este es un programa diseñado para reemplazar al Bloc de Notas de Windows y darle al usuario todo el poder y funcionalidad necesarios para satisfacer los requerimientos de edición de texto más elevados.

Para la compilación y depuración de las clases java MIDP 2.0 se han utilizado dos sistemas (ambos son equivalentes): El J2ME Wireless Toolkit y el Samsung SDK for J2ME Platform. Son semejantes, ya que los dos brindan un entorno J2ME donde poder trabajar con esta tecnología. El único motivo por el que utilicé el entorno de Samsung era para comprobar el funcionamiento de la aplicación en un móvil de dicho fabricante y, además, permitía la posibilidad de introducir multitud de paquetes opcionales (como por ejemplo el que permite hacer conexiones con FileConnection) que no aparecían en la versión de Wireless utilizada.

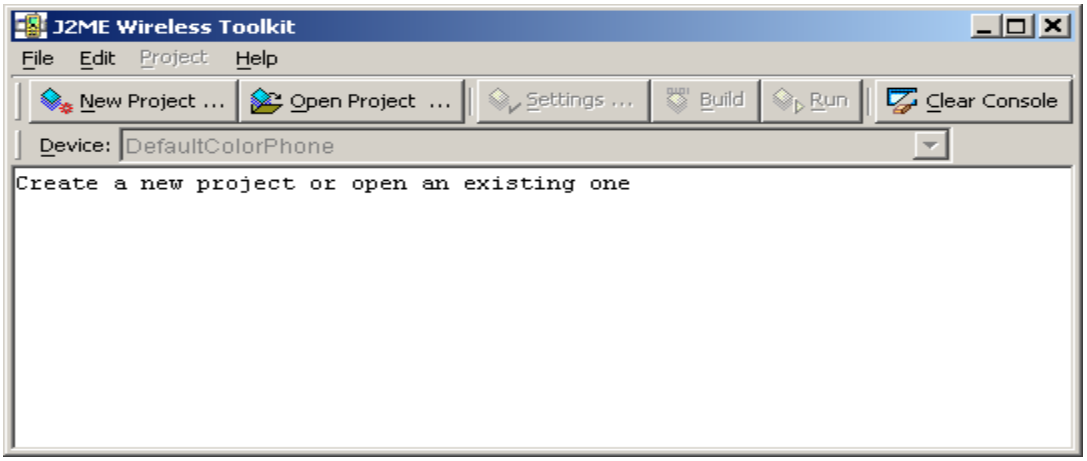
Ahora vamos a ver cómo funciona el J2ME Wireless Toolkit ([10]) a la hora de crear y trabajar con un proyecto J2ME:

KtoolBar

Lo primero de todo es bajar la aplicación, de forma gratuita, e instalarla en nuestro ordenador. Ahora buscamos el programa para empezar a trabajar.

Inicio -> Programas -> J2ME Wireless Toolkit 2.2 -> KToolbar
Aparecerá una ventana como la siguiente:

Fachada de Wireless Toolkit



Abrir un proyecto

Un proyecto está asociado a un MIDlet suite. El proyecto contiene los ficheros fuentes, los binarios y otros recursos asociados al MIDlet suite, así como el fichero JAD (descriptor) y el manifiesto. Cuando se crea un nuevo proyecto, *nombre_proyecto*, los

ficheros asociados se crean en el directorio *apps/nombre_proyecto*, con la siguiente estructura de directorios:

Src: contiene los archivos fuente.

Res: contiene recursos asociados con el MIDlet.

Bin: contiene el Jar, JAD y manifiesto.

Lib: contiene librerías externas en formato JAR o ZIP.

Para abrir un proyecto ya existente debemos seleccionar en el menú **File -> Open Project** o hacer clic sobre **Open Project**, nos aparecerá la siguiente ventana:

En ella vemos los proyectos que se incluyen con la instalación del J2ME Wireless Toolkit, seleccionamos el que nos interese abrir y hacemos clic sobre el botón **Open Project**. En la consola de ktoolbar nos aparecerá el mensaje de **Project "el_abierto" loaded**. A partir de este momento será el proyecto con el que estaremos trabajando.

Propiedades del MIDlet suite

Tanto el fichero JAD, como el manifiesto se construyen a partir de las propiedades o atributos del MIDlet suite. Desde el J2ME Wireless Toolkit, podemos ver, modificar y añadir nuevas propiedades. Para ello desde el menú **Project -> Settings...** o haciendo clic en **Settings...**. En las diferentes lengüetas aparecen los atributos obligatorios (*Required)*, los opcionales (*Optional)* y, también, los definidos por el usuario, *User Defined*, (hay que recordar que estos atributos se incluyen en el JAD y su nombre no empieza por "MIDlet-"). En la última lengüeta *MIDlets* se indican los MIDlets incluidos en el MIDlet suite, que corresponden con los que nosotros hayamos incluido en nuestra aplicación final.

Compilar un proyecto

Desde el J2ME Wireless Toolkit se puede compilar y pre verificar todos los MIDlets incluidos en el MIDlet suite que estamos desarrollando, así como crear los ficheros JAD y JAR correspondientes. Para ello ir al menú **Project -> Build** o hacer clic en el botón **Build**.

Ejecutar en un dispositivo

Por último, ejecutaremos este MIDlet suite en un emulador de un dispositivo, el J2ME Wireless Toolkit nos proporciona cinco emuladores: DefaultColorPhone, DefaultGrayPhone, MediaControlSkin y QwertyDevice, que podemos seleccionar desde la pantalla principal en la lista de selección de **Device:**

Elegimos el emulador que mejor nos parezca y ejecutamos la aplicación, para ello seleccionar en el menú **Project -> Run** o hacer clic en **Run**.

NOTA: Debemos recordar que es un emulador de un teléfono y nuestra mano es el ratón.

Ahora ya sabemos qué es lo que podemos hacer con este entorno, Ahora vamos a ver cómo empezar desde cero.

Creación del proyecto HelloWorld

Tenemos que hacer clic en **New Project** y aparecerá una ventana como la de la imagen, en la que introduciréis el nombre de proyecto, “nombre_proyecto” y el nombre del MIDlet, “nombreDelMidlet”.

Al pulsar **Create Project**, en la consola del KToolBar aparecerán mensajes indicando donde debéis almacenar los ficheros de la aplicación.

Este es el momento en el que debemos crear el código de nuestro MIDlet de manera ordenada y apropiada. Una vez que cumplimos estos objetivos, debemos colocar los archivos finalmente creados en el directorio de fuentes correspondiente del proyecto (apps/src/ nombreDelMIDlet) y compilarlos. Hay que recordar que la parte de compilación, que desde el Wireless Toolkit se realiza con sólo apretar un botón involucra los siguientes pasos:

1. Compilación.
2. Pre verificación.
3. Creación del manifiesto MANIFEST.MF (<directorio_local_aplicaciones>/nombre del MIDlet/bin/MANIFEST.MF)
4. MIDlet-1: nombreDelMIDlet, nombreDelMIDlet.png, nombreDelMIDlet
5. MIDlet-Name: nombreDelMIDlet
6. MIDlet-Vendor: Computacion en Red
7. MIDlet-Version: 1.0
8. MicroEdition-Configuration: CLDC-1.0
9. MicroEdition-Profile: MIDP-1.0
10. Creación del fichero JAD (<directorio_local_aplicaciones>/ nombreDelMIDlet /bin/ nombreDelMIDlet.JAD)
11. MIDlet-1: nombreDelMIDlet, nombreDelMIDlet.png, nombreDelMIDlet
12. MIDlet-Jar-Size: 1265
13. MIDlet-Jar-URL: nombreDelMIDlet.jar
14. MIDlet-Name: nombreDelMIDlet
15. MIDlet-Vendor: Computacion en Red
16. MIDlet-Version: 1.0

IMPORTANTE: Es necesario actualizar en los **Settings...** la propiedad **MIDlet-Jar-Size** al tamaño correspondiente y saber que hay otros atributos que en QR-APDID cambian de valor como, por ejemplo, MicroEdition-Profile: MIDP-2.0

Con esto, ya podríamos hacer los pasos antes descritos: compilación, búsqueda de emulador y ejecución.

El Samsung SDK for J2ME Platform tiene un funcionamiento muy semejante, por eso no vamos a ver su funcionamiento puesto que con el descrito para Wireless Toolkit es más que suficiente para poder utilizar este otro entorno de desarrollo.

Con todo esto tenemos todo el apartado del Estado del Arte. Ahora vamos a ver el funcionamiento de QR-APDID a alto nivel para, posteriormente, ver su funcionamiento a más bajo nivel.

Capítulo 3: Descripción a alto nivel de QR-APDID

QR-APDID es una sencilla aplicación móvil que permite extraer información de un CQR a partir de una foto a dicho código. La aplicación brinda una nueva manera de obtención de información de manera segura y ordenada, orientando y ayudando al usuario dentro del contexto en que se encuentra.

Primero vamos a explicar los elementos clave en el contexto de QR-APDID: Hay dos tipos de personas en esta aplicación con roles diferenciados. Uno es el administrador o administradores y el otro es el usuario o cliente.

1.- Administrador: El administrador o administradores es la persona o personas que "contratan" el servicio de QR-APDID para establecer una red de códigos en el establecimiento o lugar donde el administrador tiene las máximas competencias. El administrador recibe este nombre porque es el que decide qué información es la que quiere brindar a aquéllos que utilicen sus terminales móviles para adquirirla. Por ello, esta es la persona que establece el juego de CQR en el lugar en el que se encuentre y dentro del contexto en el que se está trabajando.

2.- Usuario: El usuario o cliente es la persona que llega a un lugar determinado y para adquirir cierto tipo de información hace uso de su terminal móvil con QR-APDID previamente instalado. Su única función es la de realizar una captura a un CQR y seguir los sencillos pasos que definen a esta aplicación, de manera que obtiene la información detallada y exacta que el administrador quiere que sepa en un par de "clics". La obtención de los resultados se hace de forma cómoda y segura en la pantalla del terminal móvil del usuario. La diferenciación entre usuario y cliente es debido al ámbito que se da a QR-APDID. O sea, que dependiendo del contexto en el que nos encontremos, a la persona que realiza la captura se le llama de una manera u otra. Es cliente cuando, por ejemplo, esté en un ámbito de tipo comercial (una tienda de ropa, de electrodomésticos, etc...) y es usuario cuando, por ejemplo, el lugar sea de espacio abierto y los CRQ tengan una finalidad de tipo totalmente orientativo (por ejemplo, la montaña).

Con todo esto, ahora pasamos a ver qué es esta aplicación para unos y para otros.

QR-APDID, a nivel de administrador, es la mejor forma de servir la información a los usuarios/clientes de forma rápida y segura. Él se tiene que encargar de saber qué es lo que quiere servir, en cuanta capacidad y los posibles destinatarios. Con QR-APDID se consigue un entorno de interacción que incentiva a los usuarios a utilizarlo para conseguir los mejores resultados en el ámbito en el que se encuentren. Lo principal es que se haga una buena difusión de este servicio para que los resultados sean los mejores posibles. El administrador debe establecer el juego de CRQ que desea poner en los lugares que desee y poner cierta información adicional para que los usuarios/clientes sepan qué es lo que desean buscar. Además de todo esto, el administrador debe dar una serie de posibilidades con las que cuenta al utilizar esta aplicación, y es la forma en que sirve la información. La información servida se puede almacenar de tres formas distintas: directamente en el mismo CQR, de forma que la

información es obtenida directamente del mismo a través de la captura (de esta forma se pueden obtener imágenes en formato .png, texto y test de prueba para verificar los conocimientos adquiridos por el usuario en la materia específica); otra forma de guardado es la específica de los terminales móviles, es decir, servir un paquete comprimido con todos los archivos necesarios para que el usuario pueda interactuar plenamente con el entorno sin necesidad de adquirir la información de otro modo que no sea accediendo a su propio teléfono (en este caso, el administrador debe elegir la forma en que sirva los archivos a los usuarios para que los mismos guarden la información en sus terminales y, una vez en el sitio apropiado, utilicen QR-APDID para empezar a obtener la información de manera ordenada y apropiada), por lo que se puede llegar a la conclusión de que todo el entorno de interacción se aloja en nuestro terminal móvil; finalmente, la otra opción de almacenamiento es la de alojar la información en un servidor en el que se guardan todo los archivos necesarios. Esta última opción es igual que la segunda, solo que los datos no se encuentran en el terminal móvil, sino en un servidor, y que la manera de acceder a los mismos es mediante una conexión a internet. Por ello, en un apartado anterior vimos que la opción de acceso a servidor sería óptima si viniera acompañada de la opción de utilización de archivos alojados en el propio terminal porque hay terminales móviles que solo pueden acceder a internet con contrato de datos y, obviamente, no es una buena idea pagar para utilizar este servicio pudiendo hacerlo gratuitamente utilizando la opción dos. De todos modos depende del administrador la forma en que sirve la información, por lo que solo podemos ver los mejores puntos para crear un entorno dinámico con fuerte éxito futuro. Las dos últimas opciones son creadas para dar al administrador la posibilidad de servir información multimedia (una información incapaz de encapsular directamente en un CQR) por lo que su uso es totalmente opcional para el administrador ya que según lo que quiera mostrar, estas opciones tendrán sentido utilizarlas o no. Este sentido debe ser estudiado por el administrador. Por ejemplo, si en el entorno no se quiere utilizar multimedia, se debería de estudiar si merece la pena establecer servidores para mostrar información que se puede mostrar sin necesidad del mismo (un caso en el que se podría utilizar es cuando el administrador prevé que en un futuro la información evolucionará a otro tipo) o si hay necesidad de mostrar la información a partir de los datos previamente bajados al terminal móvil. En definitiva, estudiar las viabilidades económicas y el rendimiento de la utilización de QR-APDID. Con todo esto, el administrador concretará todos los parámetros necesarios y establecerá todas las formas de almacenamiento y difusión de la información para establecer esa realidad aumentada en el ámbito estudiado.

Ahora veremos QR-APDID a través de los “ojos” del cliente/usuario: QR-APDID a nivel de usuario es una sencilla aplicación que permite adquirir conocimientos acerca de productos, alojamientos, destinos, caminos, etc... Esta aplicación consigue unificar varios tipos de mensajes e información que ayudan al usuario a adquirir los conocimientos de manera bastante sencilla. La aplicación consiste, sencillamente, en hacer una captura de un CQR, observar qué mensaje aparece (este es un mensaje introductorio del tipo de información que estamos observando) y aceptar la visualización del mismo. Estos mensajes que se pueden obtener son de varios tipos:

Tipo 1: Texto – Nos aparecerá en el terminal un mensaje, después de haber realizado la captura, diciéndonos que estamos intentando observar un tipo de información que consiste en texto codificado (caracteres Ascii). Los textos pueden venir codificados en el mismo CQR o pueden encontrarse tanto en el servidor (archivo que

contiene el texto que, posteriormente, nos aparecerá en el terminal) o en nuestro terminal.

Tipo 2: Imagen – Al igual que el texto, en la pantalla aparecerá un mensaje que nos avisa de que vamos a intentar visualizar una imagen (el formato de las imágenes utilizado es .png). La codificación de las imágenes depende de la forma en que se haya adquirido la misma (más adelante veremos cuáles son las formas de adquisición de estas imágenes). La forma de adquirir dicha información puede ser de cualquiera de las tres que QR-APDID permite como formas de almacenamiento de la información.

Tipo 3: Audio – QR-APDID nos avisa, una vez hecha la captura de un CQR, de que estamos a punto de reproducir un archivo mp3, por lo que estamos ante la posibilidad de poder escuchar la información. La adquisición de estos archivos puede ser o a través del servidor o a través de un archivo dentro de nuestro propio terminal.

Tipo 4: Vídeo – Misma tónica que con los audios pero con vídeos. Los archivos utilizados son de tipo mp4 y la forma de gestión y almacenamiento son las mismas que las detalladas para mp3.

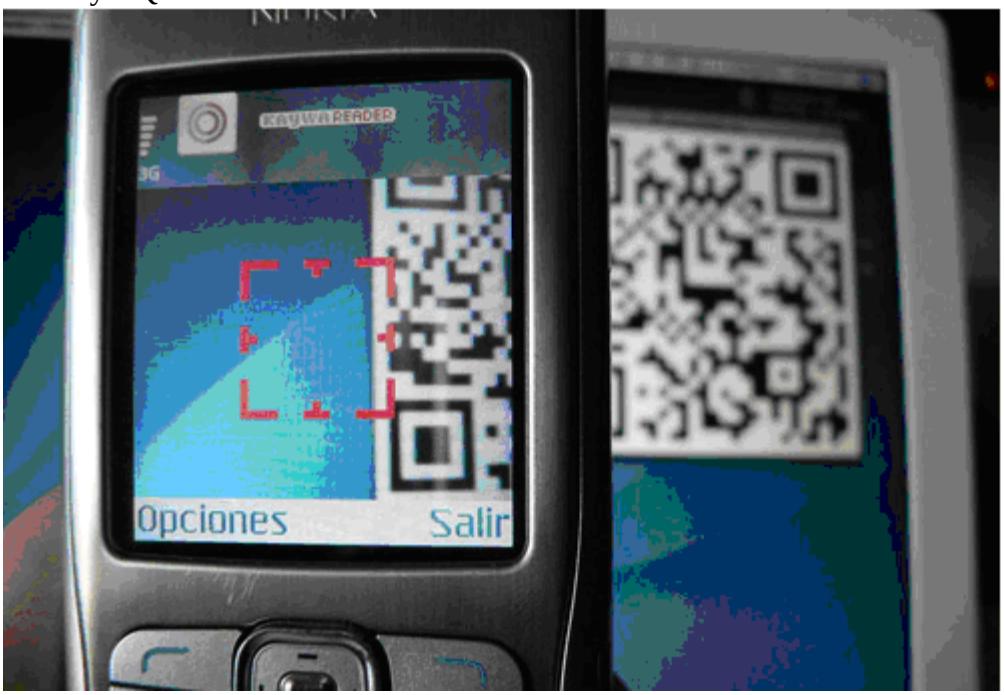
Tipo 5: Test – Este es un tipo especial en el que no se persigue el objetivo de informar, sino que la finalidad es la de permitir al usuario/cliente conocer su nivel de conocimiento a partir de una pregunta, ofreciendo varias posibles respuestas entre las cuáles se encuentra la respuesta correcta. Con ello se consigue que la persona aprenda y sepa si ha comprendido todos los datos necesarios para tomar decisiones o saber orientarse en un ámbito concreto. La forma de servir este tipo de información es solamente a partir de la codificación propia en el mismo CQR.

Después del mensaje que nos indica qué tipo de información vamos a ver, procedemos a visualizar la información.

Para poder utilizar las partes opcionales, es decir, el acceso a servidor o la posibilidad de visualizar la información a partir de archivos que se alojan en nuestro propio terminal, debemos tener conocimiento de los servicios que presta el administrador que ha instaurado QR-APDID y saber si las opciones son interesantes de utilizar desde el punto de vista del terminal móvil o no. Si tenemos un terminal en el que no tenemos conexiones wifi y que, además, no disponemos de ningún tipo de conexión de datos (tarifas planas) es interesante darse cuenta de que el servicio de adquisición de información vía conexión no es el apropiado para ser utilizado con nuestro terminal. Por ello sería mejor utilizar la opción de descargar los archivos en nuestro teléfono y, a partir de ahí trabajar con ello. Otra restricción puede ser nuestra memoria del teléfono, para lo cual el administrador debe saber que los archivos que quiera servir a sus clientes/usuarios deben ser de unos tamaños determinados o realizar una división de archivos según el tema tratado, sección específica, etc... (Todo esto son competencias que el administrador posee, que debe estudiar y que tiene que tener en cuenta a la hora de la toma de estas decisiones). Bien es verdad que los terminales móviles de hoy en día no necesitan ser excesivamente potentes como para que esta última posible restricción sea una realidad, pero como aún a día de hoy en la calle siguen existiendo, es importante que los administradores tengan en cuenta estos hechos para que QR-APDID pueda llegar al máximo número de clientes o usuarios.

Después de realizar la visualización, sencillamente se da al botón de “captura” y, nuevamente, estamos en la pantalla de captura de imagen en la que podremos volver a realizar una foto de un nuevo CQR.

Nokia y CQR



Con un “clic podemos disponer de toda la información necesaria y aquélla que el administrador cree que necesitamos para poder interactuar dentro del espacio que necesitamos.

Como bien hemos apuntado en apartados anteriores, los ámbitos en que QR-APDID puede ser manejado pueden ser ampliados. Es decir, que no solo para centros comerciales o lugares públicos, sino que también puede ser usado en centros de ventas de coches, centros de ocio, restaurantes, etc...

Gracias a la gran capacidad de información que se puede codificar directamente en un CQR ya solo queda a la disposición de la cámara con la que se realiza la fotografía la capacidad de descodificar imágenes o texto, que son los dos tipos de información que pueden llegar a ocupar grandes sumas de caracteres en el mismo CQR (las preguntas tipo test no tendrían por qué ser muy largas pero todo ello queda a gusto de lo que el administrador desee). Así que este también debe ser otro de los puntos reseñables para que el administrador cree un entorno óptimo de interactuación para los usuarios.

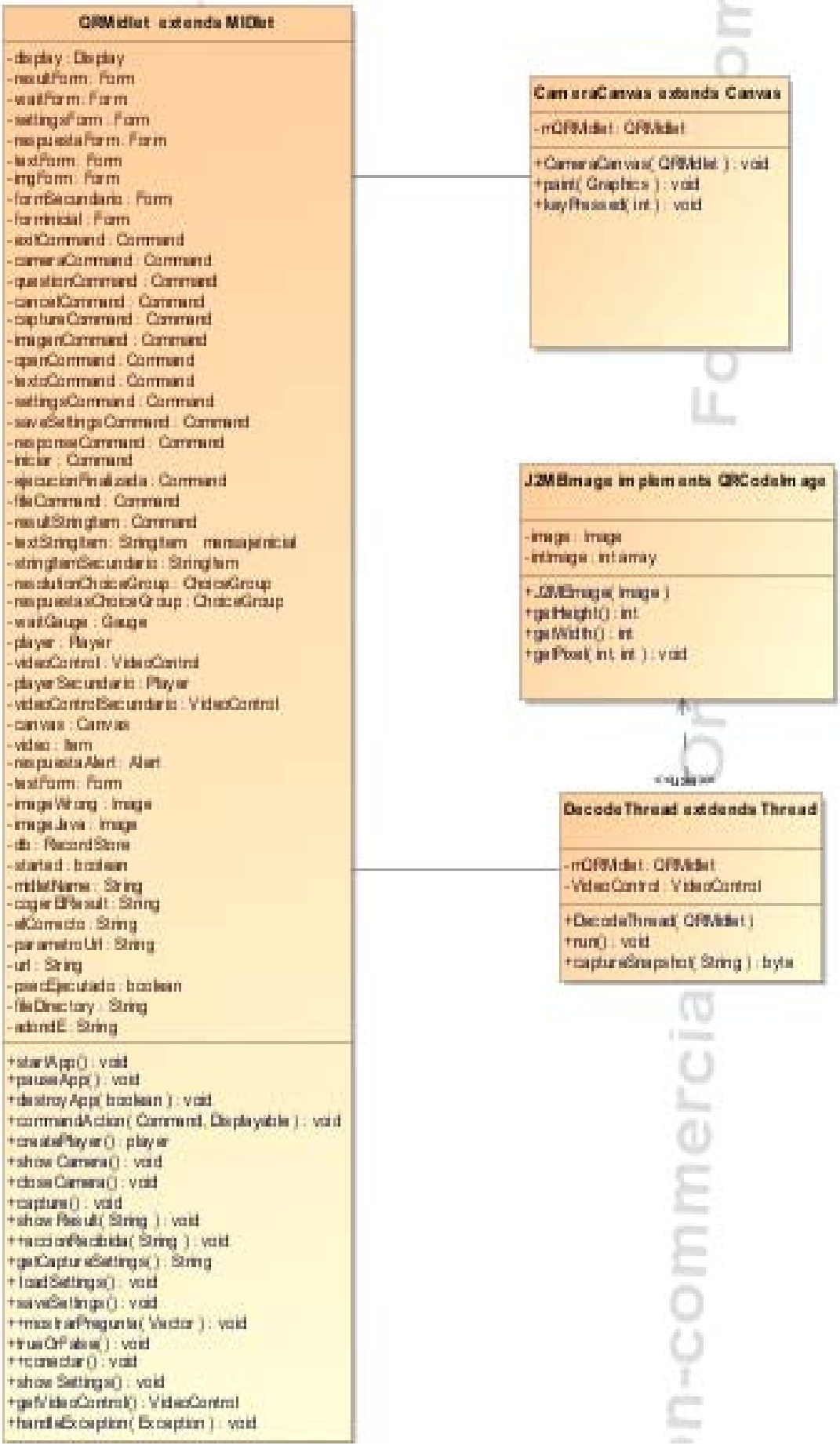
Visto todo esto, cabe destacar varios puntos: el administrador debe crear un entorno donde se estudie seriamente la viabilidad económica a partir de los tipos de información que quiera servir; se deben establecer los parámetros necesarios para encontrar un "punto medio" en el que la gran mayoría de los usuarios puedan usar sus terminales móviles sin ningún problema para incentivar el mayor uso posible de QR-APDID (esto consiste en tener en cuenta todas las restricciones posibles y todos los elementos necesarios para hacer de esta aplicación un arma potente); y el cliente o usuario debe saber cuál es la mejor manera de utilizar QR-APDID teniendo en cuenta las posibilidades que se le ofrece dentro de un marco donde el administrador es el encargado de dar las diferentes oportunidades. Por todo esto, el administrador debe saber también que lo mejor es ofrecer todos los posibles servicios si sirve todos los tipos de información posibles (En el caso de no servir multimedia, claramente no debería de utilizar las opciones de conexión de internet o conexión interna del terminal).

Una vez hecha una descripción a alto nivel de QR-aPDID, donde hemos visto cómo se ve la aplicación desde el punto de vista de administrador y desde el punto de vista de ususario/cliente, donde también hemos visto detalles para su buena

implementación y detalles de su uso, ahora veremos, detalladamente, su funcionalidad a bajo nivel.

3.1 Diagrama de Clases

En este apartado vamos a ver la relación de clases de QR-APDID, donde vamos a modelar solamente las clases principales y daremos esbozos de sus relaciones y los usos de las librerías. Las clases son: QRMidlet, CameraCanvas, J2MEImage y DecodeThread. Las demás clases de la aplicación pertenecen a esas librerías que ya hemos avisado que permiten el soporte de descodificación de CQR y que solo vamos a referenciar su uso pero no vamos a dar definición alguna de ellas:

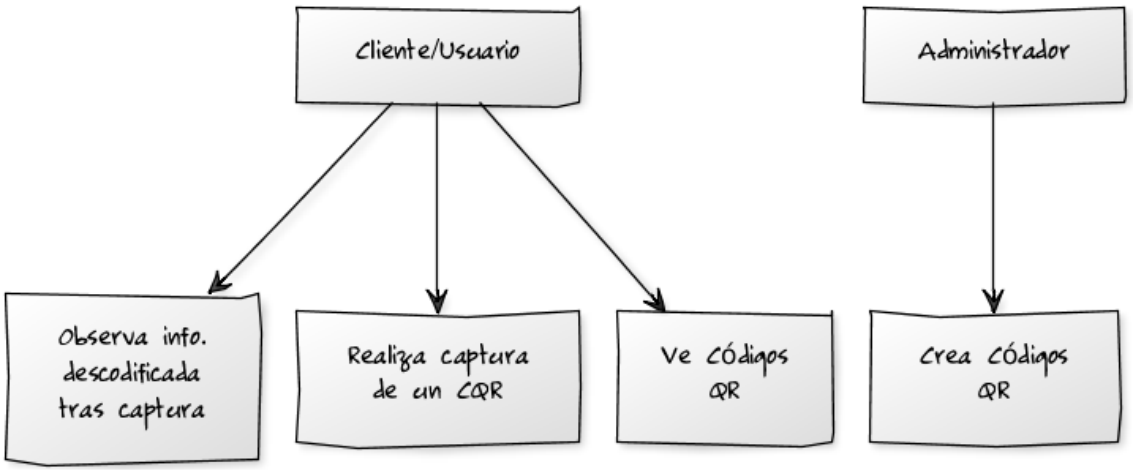

QRMidlet extends MIDlet
-display : Display
-resultForm: Form
-waitForm: Form
-settingsForm : Form
-respuestaForm: Form
-textForm: Form
-imgForm: Form
-formSecundario : Form
-formInicial : Form
-exitCommand : Command
-cameraCommand : Command
-questionCommand : Command
-cancelCommand : Command
-capturaCommand : Command
-imagenCommand : Command
-openCommand : Command
-textoCommand : Command
-settingsCommand : Command
-saveSettingsCommand : Command
-responseCommand : Command
-iniciar : Command
-ejecucionFinalizada : Command
-fileCommand : Command
-resultStringItem : Command
-textStringItem : StringItem mensajeInicial
-stringItemSecundario : StringItem
-resolutionChoiceGroup : ChoiceGroup
-respuestasChoiceGroup : ChoiceGroup
-waitGauge : Gauge
-player : Player
-videoControl : VideoControl
-playerSecundario : Player
-videoControlSecundario : VideoControl
-canvas : Canvas
-video : Item
-respuestaAlert : Alert
-testForm: Form
-imgWrong : Image
-imgJava : Image
-db : RecordStore
-started : boolean
-midletName : String
-codigoBResult : String
-elCorrecto : String
-parametroUrl : String
-url : String
-pasoEjecutado : boolean
-fileDirectory : String
-adondE : String
+startApp() : void
+pauseApp() : void
+destroyApp(boolean) : void
+commandAction(Command, Displayable) : void
+createPlayer() : player
+showCamera() : void
+closeCamera() : void
+capture() : void
+showResult(String) : void
++accionRecibida(String) : void
+getCaptureSettings() : String
+loadSettings() : void
+saveSettings() : void
++mostrarPregunta(Vector) : void
+trueOrFalse() : void
++conectar() : void
+showSettings() : void
+getVideoControl() : VideoControl
+handleException(Exception) : void
CameraCanvas extends Canvas
-mQRMidlet : QRMidlet
+CameraCanvas(QRMidlet) : void
+paint(Graphics) : void
+keyPressed(int) : void
J2MEImage implements QRCodeImage
-image : Image
-intImage : int array
+J2MEImage(Image)
+getHeight() : int
+getWidth() : int
+getPixel(int, int) : void
DecodeThread extends Thread
-mQRMidlet : QRMidlet
-VideoControl : VideoControl
+DecodeThread(QRMidlet)
+run() : void
+captureSnapshot(String) : byte

En este diagrama vemos tanto los atributos como los métodos de las cuatro clases principales de la aplicación. Desde QRMidlet (la clase principal) se llama a la clase DecodeThread cuando se realiza una captura (en el método capture()). DecodeThread crea un hilo de ejecución que convierte la captura en una imagen de QR (J2MEImage) y realiza la descodificación usando las librerías utilizadas (en el método run()). CameraCanvas es la pantalla que utilizamos para ofrecer la posibilidad de capturar una imagen.

3.2 Casos de uso

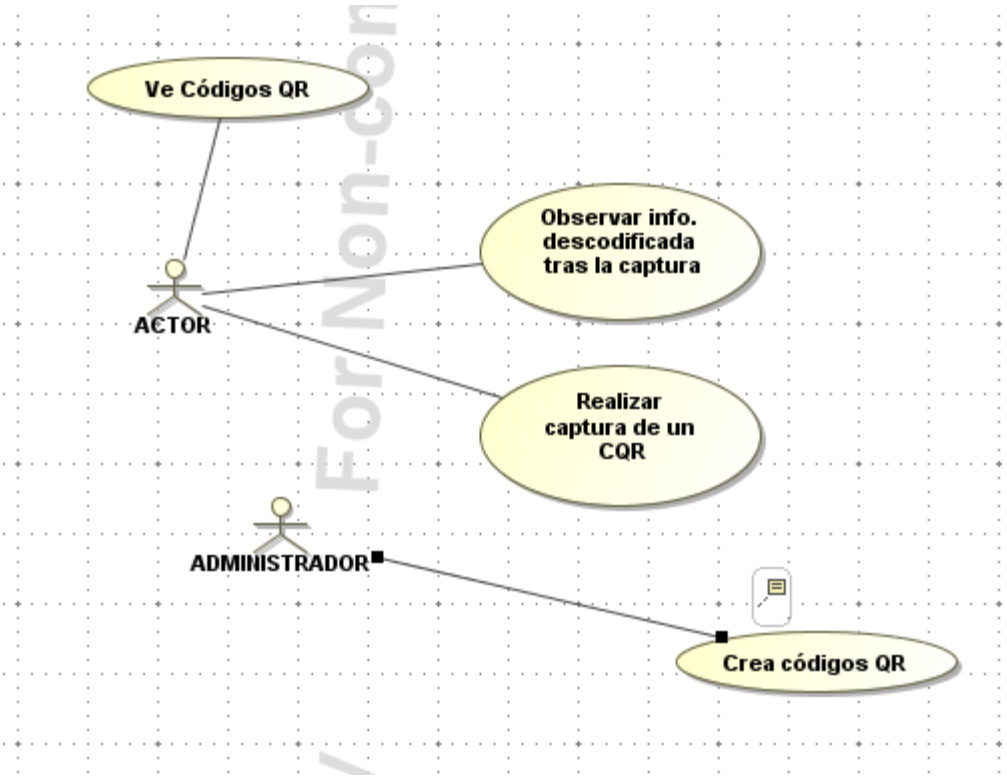
Vamos a ver, en este apartado, los casos de uso que se dan en el entorno de utilización de QR-APDID, donde tenemos dos personajes intervinientes: el administrador (se puede llamar administrador a la persona ó conjunto de personas que decide/n qué información y de qué tipo estará codificada en los CQR) y el cliente/usuario (posee un nombre distinto dependiendo del entorno en el que se encuentre).

La gráfica siguiente modela todo esto:



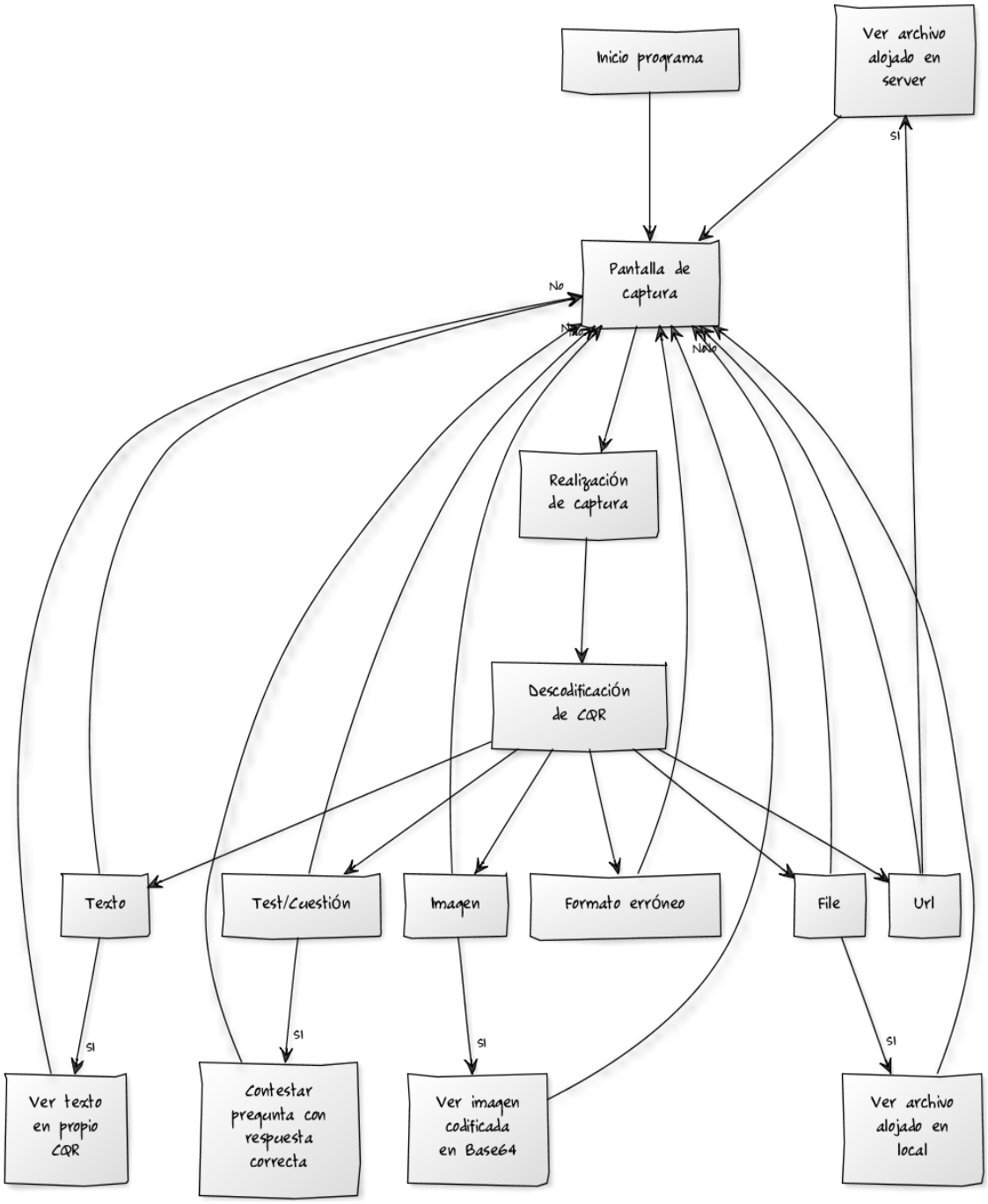
Aquí observamos las acciones que se producen desde el punto de vista de cada Personaje a la hora de interactuar en un entorno específico con QR-APDID.

Ahora vamos a ver el mismo caso con un diagrama UML:



3.3 Diagrama de Flujo

Aquí vemos el hilo de ejecución de la aplicación. El fin de la aplicación no aparece puesto que en cualquier momento de la ejecución de la aplicación se puede finalizar la misma (También hay que salvar las distancias con la manera de tratar un archivo que provenga del server, puesto que si es texto se trata con un servlet y en cualquier otro caso, no):


Inicio programa
Ver archivo alojado en server
SI
Pantalla de captura
No
No
No
Realización de captura
Descodificación de CQR
Texto
Test/Cuestión
Imagen
Formato erróneo
File
Url
SI
SI
SI
SI
Ver texto en propio CQR
Contestar pregunta con respuesta correcta
Ver imagen codificada en Base64
Ver archivo alojado en local

3.4 Parte Web

La parte web consiste en una sencilla pagína jsp en la que el administrador puede comprobar las estadísticas del uso de la aplicación en su entorno correspondiente. Es el único que puede acceder a esta página que está alojada en el servidor donde, además, se encuentran todos los archivos guardados del entorno del administrador. En adición a lo dicho, se pueden generar los códigos QR necesarios para la interactuación de los usuarios con el entorno.

En definitiva, esta página consiste en una sencilla herramienta de gestión estadística y creativa que complementa a todo el entorno de QRAPDID.

Capítulo 4: Descripción detallada

QR-APDID es el nombre de la aplicación que he realizado. Su nombre viene de **AP**licación **DID**áctica, la cual consiste en la descodificación de códigos QR para la obtención de distintos tipos de información, tales como imágenes, texto, cuestiones de tipo test, url's y obtención de archivos directamente del terminal para poder interactuar con un entorno, previamente elaborado, gracias a la información obtenida de realizar fotos con un móvil utilizando esta aplicación. Y es que ese es el funcionamiento de la aplicación: con nuestro móvil hacemos una captura a un código QR utilizando la aplicación QR-APDID, la aplicación procesa la información guardada en ese código (una serie de acciones entre las cuales están la descodificación del código y la diferenciación de la información guardada en el mismo) y nos da la posibilidad de visualizar dicha información, dentro de la misma aplicación, para poder visualizar los contenidos "disfrazados" en la información obtenida.

Los tipos de información guardados los explicaremos uno a uno en los siguientes apartados, mientras que el funcionamiento es muy simple e interesante, tal y como hemos explicado antes, ya que permite introducirnos en un entorno previamente desarrollado y elaborado donde podemos aprender y descubrir todo lo que el administrador quiera enseñarnos sin necesidad de que él mismo nos lo cuente, si no que nosotros solos, con la ayuda de QR-APDID, podemos ser capaces de aprender un montón de cosas, teniendo la posibilidad de disfrutar de multitud de servicios que nos ayuden a hacerlo de manera amena, divertida y siempre teniendo en cuenta el fin último que es aprender.

A nivel de código, QR-APDID ha sido desarrollado en Java MIDP 2.0 y utiliza una serie de clases predefinidas que nos sirven como librerías, además de una serie de clases en las que se hace uso de estas librerías, para poder implementar el lector de códigos QR (CQR) y que nos sirven como fuente inicial del código resultante, a partir del cual se han ido haciendo cambios y se han ido sumando todas las funcionalidades que QR-APDID necesita para constituirse como tal. Por lo que hay que apuntar que toda la funcionalidad de la aplicación ha sido implementada "por encima", es decir, se añade toda la funcionalidad que se requería para la aplicación y se da forma a la misma tanto de manera visual como de manera funcional de forma independiente (el lector en sí es lo único que no ha sido implementado, es decir, librerías y uso de las mismas).

Una vez visto cuál es el objetivo de QR-APDID, su uso, características y, muy por encima, cómo ha sido construido, ahora vamos a ver en detalle los tipos de información que es capaz de procesar (codificados en CQR) y examinaremos la política que se ha seguido para codificar en java dicha funcionalidad. QR-APDID es capaz de descodificar distintos tipos de información y de permitir interactuar al usuario a partir de la información que se obtiene tras la descodificación. También veremos, en cada caso, la forma en que viene codificada la información.

4.1 Información codificada

En este apartado vamos a ver cómo viene codificada la información y como son tratados los diferentes tipos por QR-APDID.

4.1.1 Texto

El primer tipo de información que estudiaremos es la de tipo texto, que consiste en una cadena de caracteres, guardada en el código QR, que componen un mensaje. Esta es la forma del mensaje codificado:

text://(texto que vamos a leer)

"text" es el identificador que se le da a la información codificada cuando queremos que QR-APDID sepa que tiene que gestionar información de tipo "texto". Cuando QR-APDID sabe que es texto lo que tiene que gestionar, coge el "texto que vamos a leer" y lo muestra en la pantalla del móvil. Así, cada vez que estemos delante de un CQR que contenga texto, podremos leer por pantalla lo que dicho CQR contiene, gracias a que QR-APDID lo descodifica.

Con este tipo de información el administrador puede proveer de información al usuario, dándole una serie de indicaciones para la correcta ejecución del ejercicio, pasos a seguir, explicaciones varias, etc. En fin, un mensaje aprovechado por el administrador para contar todo lo que crea que es necesario para el usuario dentro del ámbito en el que nos movemos.

Los textos son caracteres ascii, que más adelante hablaremos un poco de ellos, en el apartado de imágenes.

4.1.2 Preguntas tipo Test

Con las preguntas de tipo test conseguimos establecer un entorno en el que se hace una cierta autovaloración de los conocimientos aprendidos acerca del mismo. Gracias a este tipo de información, el usuario sabrá si "va por el buen camino" o no al poder valorar su nivel de conocimientos.

Al igual que el tipo "text", este tipo de información también consiste en una cadena de caracteres, pero con características especiales. Esta es la forma del mensaje codificado:

question://(pregunta)*(resp1)*(resp2)*...*(respn)*(respcorrecta)-

"question" es el identificador que se le da a este tipo de información para que QR-APDID la procese como pregunta de tipo test. La codificación de este tipo de información en Java permite coger la pregunta por un lado, las respuestas por otro y la respuesta correcta de manera sencilla y mostrarlas en pantalla para que el usuario responda lo que crea conveniente. Hay un par de normas a seguir en la codificación de preguntas tipo test: La primera es que tenemos que codificar la información siempre como hemos visto arriba, es decir, que tenemos que poner los "*" y el "-"(separadores) siempre para que el sistema consiga generar correctamente la pregunta. Un ejemplo:

question://¿Quién es tu mejor amigo?*Juan*Pedrito*Juan-

Tenemos que hacerlo siempre de la misma manera.

La segunda es que no podemos poner espacios entre el texto (pregunta o respuestas) y los separadores para que la información sea bien gestionada por la aplicación, o si no, podrían producirse errores al coger las respuestas (mal tratamiento de las cadenas de caracteres).

Con esto, conseguimos que el administrador ofrezca al usuario la posibilidad de autoevaluación y comprensión de conocimientos sin necesidad de tener a nadie al lado que se lo diga, teniendo siempre en cuenta que el fin último es el correcto aprendizaje del entorno en concreto.

Tampoco hay que olvidar que el tipo de caracteres utilizados son ascii, igual que el caso de texto ("text").

4.1.3 Imágenes

Con las imágenes, conseguimos ampliar las capacidades de QR-APDID, ya que consigue leer imágenes codificadas cuyo formato es ".png". La forma utilizada para guardar una imagen en un CQR es utilizando la codificación Base64, que es un sistema de numeración posicional que usa 64 como base. Es la mayor potencia de dos que puede ser representada usando únicamente los caracteres imprimibles de ASCII. Todas las variantes famosas que se conocen con el nombre de Base64 usan el rango de caracteres A-Z, a-z y 0-9 en este orden para los primeros 62 dígitos, pero los símbolos escogidos para los últimos dos dígitos varían considerablemente de unas a otras. La imagen ".png" es lo que se guarda codificado en Base64 en el CQR (es decir, los caracteres que la representan), de manera que cuando hacemos una foto a un CQR que contiene una imagen, QR-APDID primero descodifica el CQR para quedarse con el formato de mensaje que representa a la información de imagen y luego descodifica la imagen para mostrarla por pantalla. Este es el formato del mensaje codificado:

image://(imagen codificada en Base64)

"image" es el identificador utilizado para este tipo de información, que avisa a QR-APDID del tratamiento que tiene que darle al mismo, descodificando, posteriormente, la "imagen codificada en Base64" para mostrarla por pantalla. Esto permite al "administrador" una forma más amena y visual de transmitir conocimientos e información sobre el entorno en el que se trabaja al usuario, mientras que el mismo usuario puede ser capaz de explorar, conocer y averiguar definiciones o hechos simplemente con la visualización de una imagen.

4.1.4 URL's

Con este tipo de información se consigue que QR-APDID aumente considerablemente su capacidad computacional a nivel de información a procesar debido a la carga capaz de soportar sin necesidad de que la misma se encuentre explícitamente en el CQR. Esto es posible ya que QR-APDID se conecta con un servidor alojado en un entorno local (Apache-Tomcat) donde se encuentran archivos que pueden ser de vídeo, texto, imágenes o audio, utilizados para explicar y aumentar los conocimientos del usuario, así como para la ayuda a la comprensión del entorno por parte del mismo. Puesto que parte de estos tipos de informaciones no se pueden alojar codificados en un CQR recurrimos a las URL's para, a través de la red, descargar información en el móvil con el fin informativo que en todo momento nos compete. La URL a la que nos conectamos es la que tiene el servidor donde se alojan todos los distintos tipos de archivos (luego explicaremos la forma del Server y cómo nos conectamos a él, y lo que aloja en él), el cuál es un atributo constante de la aplicación dentro de su arquitectura, y la forma de los mensajes codificados en CQR para que APDID trate este tipo de información es la siguiente:

url://(tipo de archivo)*(nombre de archivo)-

El identificador utilizado para que QR-APDID sepa que debe trabajar con la conexión al server, para descargar un archivo y mostrarlo por la pantalla del móvil, es "url" (como ya hemos dicho, la URL del server la tenemos establecida como un atributo de la aplicación (esta aplicación es la primera versión, la 1.0), lo que nos permite bastante soltura a la hora de cambiar o quitar dicha URL sin necesidad de realizar cambios en el código de la aplicación). Una vez que QR-APDID sabe que el tipo de información codificada en CQR es "url", entonces realiza una petición (POST) con destino "URL del server" enviando un único parámetro en el que se encuentran "tipo de archivo" y "nombre de archivo" que los coge del mensaje codificado en CQR de idéntica manera a como vienen por definición, siendo el propio server, posteriormente, el que se encargue de trabajar con ellos de forma separada o reproduciendo directamente los archivos concatenando la url del server con los datos obtenidos para localizar el archivo en el servidor. El "tipo de archivo" es un solo carácter cuyo valores puede ser: "v" (vídeo), "a" (audio), "i" (inagen) o "t" (texto), lo que permite que después, en el server, se sepa en qué directorio buscar el archivo especificado por

“nombre de archivo”, que es el nombre del archivo que se desea descargar al móvil. Toda esta gestión la lleva el server mientras que en la parte del móvil, simplemente se esperan las respuestas llegadas de la petición que, después, se mostrarán por pantalla. Tanto vídeo como audio se mostrarán por pantalla gracias al uso de MMAPI (con el respectivo Player y Videocontrol) mientras que la imagen o el texto se plasmarán en pantalla para visualización normal por parte del usuario. Finalmente, comentar que la forma del mensaje codificado, al igual que en los otros casos, siempre debe ser esta para que la aplicación funcione correctamente.

Una vez explicados los tipos de información capaces de ser procesados por QR-APDID, conviene dedicarle un capítulo especial al Server al cuál se conecta, ya que es una parte fundamental de la aplicación donde aloja multitud de archivos e información que no necesariamente se encuentran en los CQR y que sin embargo permiten aumentar muchísimo las capacidades de todo el entorno de trabajo que proporciona nuestra aplicación gracias al tipo de información “url” que da la directiva de trabajo a QR-APDID para realizar una conexión al server. Dicha conexión es gestionada por un servlet, al igual que la búsqueda y la interactuación total del server y QR-APDID. Todo ello será explicado más adelante.

4.1.5 File

Hay una sutil diferencia entre utilizar esta opción y utilizar “url”, y ésta radica en que los archivos, en vez de ser cogidos de un servidor, son cogidos de la memoria del propio teléfono móvil, de manera que los archivos son alojados en la memoria y cogidos directamente para su reproducción/visualización. Por lo demás, estamos ante dos opciones prácticamente iguales, tanto desde el punto de vista del código (donde se pueden ver sutiles diferencias pero a rasgos generales se trata de una conexión de la cual se pueden sacar todos los elementos y, una vez sacados, mostrarlos de la misma manera que si se tratara de un archivo alojado en internet) como desde el punto de vista del trato que se da a los archivos alojados en el móvil y no en un server, porque se mantiene la forma en que se alojan para saber dónde se tienen que buscar los archivos. Dicho esto, vamos a ver que a rasgos generales nos encontramos ante un trato muy parecido en todos los aspectos, con ligeras salvedades:

file://(tipo de archivo)*(nombre de archivo)-

Adquiere el mismo trato por parte de la aplicación que el tipo de información de “url”. Contiene el identificador “file” para que QR-APDID sepa de qué tipo de información se trata la codificada en el CQR. Para el “tipo de archivo” se sigue la misma dinámica que para “url, con la que se diferencian los distintos tipos de información que esta aplicación es capaz de soportar (recordamos que “t” era para texto, “i” para imagen, “v” para vídeo y “a” para audio). Estos tipos de información alojados en el propio terminal móvil, capaces de soportar por QR-APDID, son los mismos que son alojados en el servidor (la única restricción visible en este aspecto y que nada tiene que ver con la aplicación es la capacidad de memoria del terminal). La aplicación coge el “nombre de archivo” y realiza una conexión interna en el terminal para hallar el

archivo. Estos archivos han de encontrarse en la memoria externa del teléfono móvil, en una tarjeta de memoria MicroSD, ya que la codificación de la aplicación está hecha para este tipo de almacenadores de información (es la versión 1.0). Una vez que se realiza la búsqueda correspondiente del archivo especificado, se muestra en pantalla de la misma manera en que mostrábamos los archivos con la otra forma (con "url").

Como podemos ver, "url" y "file" tienen formatos muy parecidos y, en términos generales, sólo se diferencian, como hemos dicho antes, en el destino en el que se encuentran los archivos. A nivel de código son escasas las diferencias y tanto es así que para la gestión de estos dos tipos de información y la conexión al destino (server ó memoria interna) se utiliza un solo método en la clase principal del MIDlet (el método conectar()).

Ahora vamos a ver cómo se guarda y se aloja la información tanto en el servidor (antes ya hicimos referencias a él) como en el terminal móvil.

4.2 El servidor

En el servidor se alojan los archivos necesarios que el administrador quiere servir a sus clientes/usuarios. Para que la aplicación sea solvente y funcione de manera correcta y precisa se mantiene una jerarquía de archivos sencilla y dinámica para que el almacenamiento de los archivos sea estándar y accesible por parte de todos los usuarios de QR-APDID. Además de esto, vamos a explicar la forma en que se accede al servidor por parte de QR-APDID, ya que las formas de ello son distintas dependiendo del tipo de información que estemos tratando.

4.2.1 Acceso al servidor

Cuando tras hacer una fotografía a un CQR la aplicación detecta que se trata de un tipo de información etiquetada como "url", la siguiente acción que hace es la de saber de qué tipo de información se trata y, sin perder tiempo, realiza la conexión y gestiona las distintas posibilidades que existen de tener éxito (esto depende sencillamente de dos cosas, que el archivo exista o que no exista). La forma de tratar las distintos tipos de información cuando se accede al servidor es distinta, por lo que conviene hacer una distinción de las mismas:

> TEXTO – Se gestiona a través del servlet. Una vez hecha toda la descodificación y obtenido el nombre del recurso, se hace una petición al servidor con un parámetro (el nombre del fichero en formato ".txt"). Al servlet le llega esta petición, coge el nombre del archivo y lo busca dentro del directorio de los archivos del texto dentro del servidor. Una

vez que lo encuentra, escribe en salida de respuesta los datos que contiene el archivo y, a partir de aquí, es QR-APDID el que "arma" el mensaje y, posteriormente, lo muestra por pantalla...

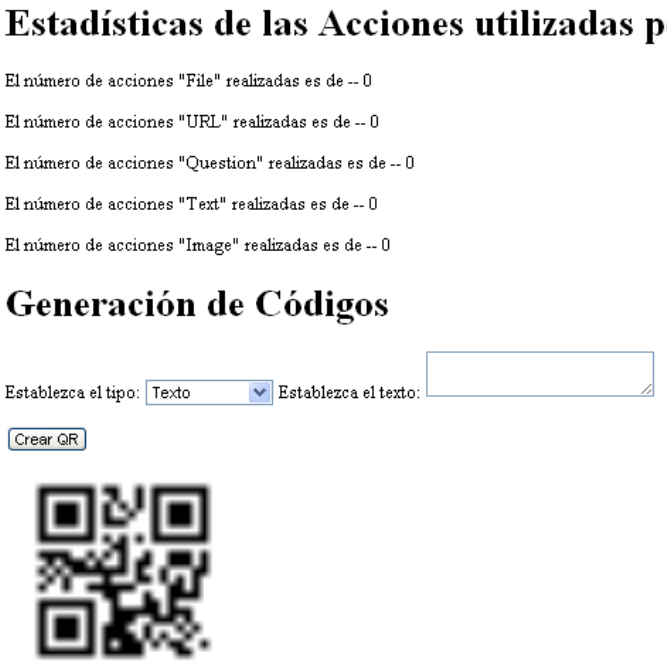
IMAGEN – QR-APDID establece una conexión con la url en la que se encuentra alojada la imagen. Esto se hace de la siguiente manera: el nombre del archivo lo coge de la descodificación del CQR (como con todos los demás), establece una conexión http con el servidor (donde la url es la del servidor y a ello le concatena el path donde se alojan las imágenes y el nombre del archivo, que es ".png") y crea una imagen directamente a partir de la respuesta recibida, que es un "inputstream", y la muestra en pantalla. Con ello está gestionada la aparición de errores.

VIDEO – Se gestiona de forma distinta a los anteriores puesto que MMAPI da la posibilidad de crear un player directamente a partir de la url donde se encuentre alojado el archivo. Como siempre se coge el nombre del archivo y, sin pensarlo dos veces, se crea el player a partir de la url donde se aloja (al igual que con imágenes, esto nos permite manejar los posibles errores que se puedan producir), se establecen los parámetros necesarios para su correcta reproducción, así como la pantalla para visualizar el vídeo, y se comienza a reproducir el mismo en pantalla.

AUDIO – Este caso es semejante al anterior. Una vez que se obtiene el nombre del archivo en el CQR, se realiza la creación del player a partir de la url en la que se encuentra. Esta url se concatena con: la url del servidor, el directorio donde se encuentra el archivo, dependiendo de qué tipo de archivo sea, y el nombre del archivo. La diferencia entre este tipo y el anterior son algunos de los elementos de inicialización que, evidentemente, no pueden ser totalmente iguales para audio que para vídeo y la url, ya que audio y vídeo se encuentran en distintos directorios dentro del servidor.

Además de esto, se dispone de una sencilla aplicación web, alojada en el servidor, que permite al administrador controlar las estadísticas donde se refleja el uso de QRAPDID por parte de los usuarios y donde se pueden generar códigos QR para su porterior uso por parte de los usuarios.

A continuación, una foto de su interfaz gráfica:



Como podemos comprobar, la interfaz gráfica posee una estructura muy sencilla: la primera parte de la misma, donde aparecen los contadores con el número de veces que se utilizó una acción de la aplicación (estadísticos). Estos números se van incrementando a medida que QRAPDID se va usando con éxito.Y la segunda parte posee un selector (donde decimos de qué tipo de información queremos crear el código QR) y un área de texto (lugar en el que ponemos el mensaje a codificar) para, una vez que se le da al botón “Crear QR”, obtener nuestro código en perfectas condiciones.

La temática es muy sencilla a bajo nivel: cada vez que se obtiene, con éxito, la información codificada de un código QR, mediante un servlet se incrementa el contador que hace referencia a la acción registrada (esto se verá reflejado en la interfaz gráfica de manera actualizada); de la misma manera se actúa con la creación de un código QR ya que, al pulsar el botón de “Crear QR”, mediante un servlet se crea la imagen en un directorio dentro del servidor a la cuál hace referencia la etiqueta del jsp en la página principal. Así de sencillo obtenemos una herramienta totalmente fácil de manipular, que complementa a todo lo visto de QRPADID.

Ahora vamos a hablar de lo referente al alojamiento de la información en el terminal.

4.3 Alojamiento en el terminal

Al igual que con el servidor, se ha establecido un modo de funcionamiento para la opción de poder ver los archivos guardados en el terminal con algunas diferencias con respecto al nombrado antes. Los tipos de información manejados son los mismos, por lo que en este aspecto no hay ningún tipo de duda. Lo que cambia es la forma en que se deben guardar los archivos en comparación con el servidor.

Vamos a ver, a parte, el tratamiento de los distintos tipos de archivos por parte de QR-APDID

4.3.1 Acceso a la tarjeta MicroSD

En la versión más inicial de QR-APDID se utilizaba la memoria del propio móvil, pero la restricción que podía significar el mismo dio qué pensar sobre la forma de guardar los archivos. Desde entonces se estudió cuál era la mejor alternativa a este hecho, y esa es la tarjeta MicroSD. La MicroSD es una tarjeta de almacenamiento de memoria flash, ésta es de las más usadas por los fabricantes para sacar sus dispositivos de memoria crítica al mercado. Con todo ello, se puede decir que de todas las memorias existentes, MicroSD es la que "engloba" a un mayor número de dispositivos y, por tanto, la que puede ser común a un mayor número de usuarios. Las hay de diversos tamaños como, por ejemplo:

Cuando QR-APDID detecta, tras la captura de una imagen, que la información guardada en el CQR es de tipo "file", coge el nombre del archivo (igual que como lo

hacía para “url”) y establece una conexión de tipo File que es distinta dependiendo del tipo de información que se esté requiriendo. A continuación vemos las diferencias:

TEXTO – Cuando se comprueba que es texto lo que hay que sacar de la memoria, QR-APDID hace una conexión de tipo FileConnection, con uri la dirección del archivo en local. Mediante un flujo de InputStream se obtienen los datos que, posteriormente a su recepción, se muestran en pantalla.

IMAGEN – Cuando se ve que el archivo pedido es un “.png”, se realiza la misma conexión que se hace con los “.txt.” y se crea la imagen con el InputStream que se recibe directamente. Después de crearlo, se muestra por pantalla.

AUDIO – Es exactamente igual la dinámica que se sigue con estos archivos “.mp3” para “file” que para “url”, solo que la única diferencia existente radica en la url con la que se crea el player, que en vez de ser la del servidor es la de la dirección local de la MicroSD, concatenada con el nombre del archivo.

VIDEO – Al igual que ocurre con el audio, ocurre con el vídeo. Una sutil diferencia existente para el caso de “url” y este caso es la url con la que se crea el player que, en este caso, es la dirección de la MicroSD en local y el nombre del archivo.

El acceso a la MicroSD por parte de QR-APDID se realiza mediante una sentencia que permite acceder a los directorios donde se alojan todos los archivos que un usuario tenga en su tarjeta.

4.4 Descodificación de un CQR

De manera totalmente independiente a todo lo que hemos contado antes, ahora vamos a contar los pasos seguidos por QR-APDID para la descodificación de un CQR. Hay que tener en cuenta que esta acción se produce sea cual sea la información que el CQR guarde en su interior, puesto que esto ocurre justo después de tomar la fotografía o captura y antes justo de descifrar la información que contiene. Aquí vienen los pasos seguidos para tal efecto:

Justo después del momento en el que se pulsa el botón de captura para realizar la fotografía, se produce una cadena de acciones que comienzan con la aparición de una “ventana de espera“para que el usuario espere mientras se producen los actos necesarios para la descodificación; a la vez, se arranca un hilo de ejecución (clase “DecodeThread”) en el que se va a recoger el resultado de la descodificación. En él se recogen los bytes de la captura, se cierra la cámara y se procede con la creación de la imagen capturada. Posteriormente, se llama al descodificador (clase QRCodeDecoder) y

se utiliza su método de descodificación pasando la imagen como argumento. En este instante se crea con la imagen, una QRCodeImagen, que es lo que gestiona el QRCodeDecoder para procesar la descodificación.

Esta descodificación consiste en una combinación de puntos en los que se va haciendo una comparativa de los mismos con la imagen capturada (puntos adjuntos guardados en un vector). Este resultado se devuelve en un array de bytes que son tratados de forma independiente tras ser comprobados todas las partes de la captura (evidentemente, se tratan todos los posibles errores resultantes de esta descodificación, ya que pueden suceder errores del tipo de que no sea un CQR lo que estemos comprobando o que este no tenga ninguna información guardada o incluso que la descodificación no haya podido ser completada).

Con ese array de bytes se crea un objeto de tipo string que es el resultado final de la descodificación, con el que la aplicación QR-APDID observa sus primeros caracteres y ya sabe qué mostrar al usuario/cliente (además ya sabe qué acciones ofrecer para que el cliente/usuario pueda interactuar con la aplicación y el entorno que le rodea).

La explicación de esta acción no es minuciosa puesto que se trata de unos actos realizados en las librerías descargadas para el soporte de descodificación de CQR por parte de este MIDlet, por lo que nos interesa que funcione, básicamente, y que haga bien su cometido. Esta serie de librerías consiste en un conjunto amplio de clases que modela cosas tales como punto, imagen de CQR, descodificador de CQR, lector de CQR, color, axis, etc... Todo lo necesario para la implementación de esta posibilidad de descodificación.

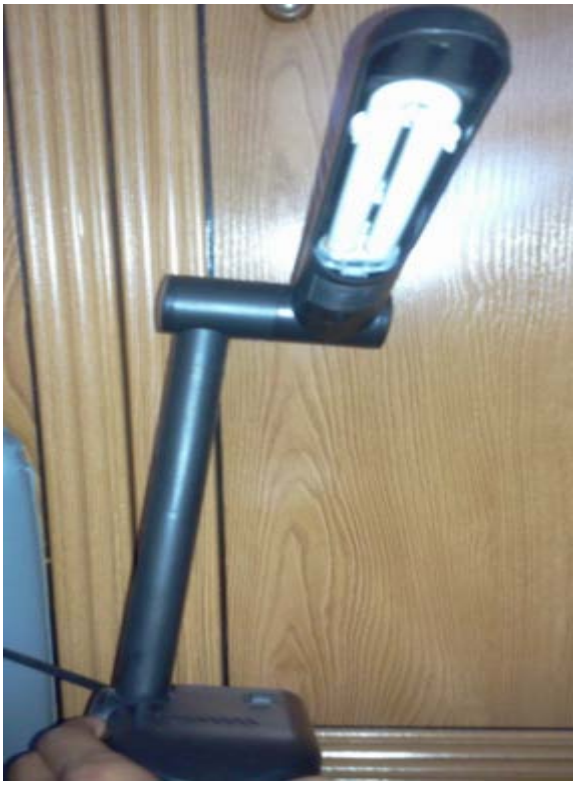
Ya hemos definido todos los aspectos fundamentales para entender de qué se trata QR-APDID tanto a alto nivel como a bajo nivel, desde lo más general a lo más específico. Así que ahora nos vamos a encargar de hacer pruebas con un par de entornos totalmente aleatorios y vamos a estudiar el nivel de aceptación de esta aplicación por parte de personas totalmente anónimas para estudiar el posible impacto de esta aplicación a nivel de usuarios/clientes.

Capítulo 5: Pruebas

Para realizar las pruebas para ver la aceptación a nivel de usuarios y el correcto funcionamiento de la aplicación, vamos a crear dos entornos distintos de ejemplo y los vamos a probar con 5 personas distintas para ver cuál es la aceptación de los mismos a esta aplicación (quizá sean pocos para llegar a una conclusión de cuál sería su aceptación a nivel más general, pero serán datos interesantes para concretar si se trata de un fracaso o no).

5.1 Entorno I

En este primer entorno disponemos de una lámpara (supongamos durante este ejemplo que estamos en el establecimiento y que la lámpara la vemos en la realidad) que estamos a punto de comprar, pero estamos dudando de su calidad, material y si realmente vale la pena comprar este modelo u otro. Como podemos comprobar, este es un ejemplo en un entorno comercial, donde el administrador quiere mostrar toda la información posible a través de los CQR. Vamos a ver el modelo de lámpara y todos los CQR que, se suponen, están al lado de este producto.



5.1.1 CQRs del entorno I

Uno de texto (explicativo),

Una pregunta tipo test,

Otra pregunta tipo test,

Un archivo de audio reproducido desde local (voz1.amr) y, por último,

Un vídeo reproducido desde el server (vid1.mp4).

Después de la explicación del siguiente entorno expondremos los resultados que, tanto este como el siguiente entorno, las personas encuestadas ofrecieron tras probar QR-APDID con estos ejemplos.

NOTA: Hemos establecido todas las posibilidades de información salvo una, la de imagen. Esta información, en este entorno, no parecía relevante, pero hay que decir que en los codificadores encontrados por internet no se ha encontrado ninguno capaz de codificar una imagen en B64 debido al gran tamaño que poseen estos mensajes. Aún así este tipo de información es soportado puesto que en pruebas realizadas a "fuerza bruta" para ver si el funcionamiento de QR-APDID era correcto se comprobó que, efectivamente, la información de tipo "image://" era correctamente soportada.

5.2 Entorno II

QR-APDID también está pensado para lugares "abiertos" como, por ejemplo, una montaña o sitios similares. En este ejemplo vamos a explicar cómo llegar a una zona céntrica de Móstoles desde una zona concreta (la estación de Móstoles Central). Por falta de tiempo no ha sido posible construir un ejemplo dentro de espacios montañosos, pero la esencia es la misma:

Nos encontramos en la estación de Móstoles Central, y entre otros, en un muro vemos información para llegar a esta conocida zona de la ciudad (es a donde queremos ir)

Texto indicativo de a donde hay que ir.

Al llegar al siguiente cruce nos encontramos con este texto y la siguiente pregunta que nos explica cómo llegar:

Aquí te dicen hacia dónde tienes que ir, por si no te has dado cuenta.

Esta es una locución donde te explican cómo llegar.

Está muy cerca este sitio, por lo que andando un poco más hacia delante ya lo encontrarías.

NOTA: Cuando se trata de sitios públicos y abiertos, quizá la posibilidad de tener servidores no sea buena, por los costes que esto conllevaría... Independientemente de esto, estas pruebas funcionan correctamente pero al pasarlas a este documento y cambiar los tamaños de los CQR parecen pervertirse las informaciones, por lo que para probar QR-APDID, lo propio es crear un entorno propio de interactuación.

5.3 ¿Qué comentan?

Todas las personas, en mayor o menor medida, vieron muy útil esta aplicación. Cuando se les explica cuál es el verdadero motivo y el fin que se quiere conseguir y, a la vez, prueban estos entornos para ver lo práctico que es este sistema, se quedan asombradas y piensan en la gran ayuda que esta aplicación podría aportar a las personas en distintos entornos.

Hay que resaltar también que todos coinciden en que esto no es más que una herramienta de "apoyo" a lo que ya existe (la información que puede ser proporcionada mediante otros medios, como otras señales o incluso personas), que no se trata de algo que pueda revolucionar al mundo, pero que consigue su fin y esto se une a la facilidad de utilización del mismo, consiguiendo así ser un mero mecanismo de información y guía para todos en un determinado momento y en un determinado lugar.

La acogida de QR-APDID ha sido positiva, en líneas generales, debido a la buena aceptación que ha habido entre las 25 personas encuestadas. Vamos a ver los resultados mediante cálculos estadísticos y datos porcentuales:

- De 25 personas encuestadas, 19 quedaron muy satisfechas, 4 quedaron satisfechas y 2 quedaron indiferentes.

  *Personas/Notas

Nota	10	9	8	7	6	5	4	3	2	1
Nº Personas	6	13		3	1		2			

  *Porcentaje

Muy satisfechas	Satisfechas	Indiferentes
76%	16%	8%

Media: 8.48 y Varianza: 2.8096

- 15 lo utilizarían siempre que fuese posible su uso, 6 lo usarían normalmente (no siempre) y 4 lo usarían rara vez (pero lo usarían).

  *Personas/Notas

Nota	10	9	8	7	6	5	4	3	2	1
Nº Personas	9	6	1	5		2	2			

  *Porcentaje

Siempre	Normalmente	Casi nunca o nunca
60%	24%	16%

Media: 8.2 y Varianza: 3.84

- 18 adquirirían esta herramienta aunque costase dinero, 4 lo pensarían (según lo que costase) y 3 no lo comprarían.

  *Personas/Notas

Nota	10	9	8	7	6	5	4	3	2	1
Nº Personas		16		4				3		

  *Porcentaje

Sin pensarlo	Pensando	No
72%	16%	12%

Media: 8.04 y Varianza: 4.1184

- 10 creen necesitar esta aplicación en su vida, 11 dudarían y 4 no creen necesitarlo.

  *Personas/Notas

Nota	10	9	8	7	6	5	4	3	2	1
Nº de Personas	5	5		7	4			3	1	

  *Porcentaje

Necesitan	Dudan	No
40%	44%	16%

Media: 7.16 y Varianza: 5.6544

- 13 personas creen que se trata de un invento muy interesante, 7 de algo interesante y 5 de un invento de poco interés.

  *Personas/Notas

Nota	10	9	8	7	6	5	4	3	2	1
Nº de Personas	7	6			7			5		

  *Porcentaje

Muy interesante	Interesante	Poco interesante
52%	28%	20%

Media: 7.24 y Varianza: 6.9024

- A 17 personas les pareció bastante novedoso, a 7 les pareció algo novedoso y a 1 no.

  *Personas/Notas

Nota	10	9	8	7	6	5	4	3	2	1
Nº de Personas		17		6	1				1	

Muy novedoso	Novedoso	No
68%	28%	4%

Media: 8.12 y Varianza: 2.5056

- A 7 personas les incentivaría a consumir más gracias a esto, 10 dudarían y 8 no.

  *Personas/Notas

Nota	10	9	8	7	6	5	4	3	2	1
Nº de Personas	6	1		5	5		5	3		

  *Porcentaje

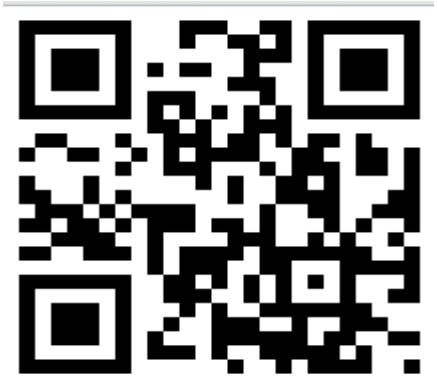
Consumen seguro	Puede que consuman	No
28%	40%	32%

Media: 6.52 y Varianza: 6.0096

Esta encuesta nos sirve para comprobar que a estas personas les ha gustado (en líneas generales). En caso de haber obtenido resultados negativos tras estas encuestas, significaría que las probabilidades de que no gustara a nadie en el mundo serían muy grandes. Ante estos resultados, se pueden tener esperanzas...

5.4 Otros ejemplos de CQRs

Estos ya no tienen que ver con entornos, sencillamente son CQRs que muestran información, anuncian, etc...



f1.mp3 cargado desde el server.

pato.mp3 desde server.

java.png desde el server.

valle.txt desde el server.

wrong.png desde MicroSD.

aaa.txt desde MicroSD.

Estos dos últimos CQR son de tipo “text://”.

Después de haber visto ejemplos de entornos determinados y otros ejemplos de aplicación de QR-APDID, ahora vamos a concretar las conclusiones finales de este proyecto y posibles mejoras futuras del mismo.

Capítulo 6: Conclusiones y mejoras del proyecto

Tras haber hecho las pruebas descritas antes y haber concluido con las respuestas afirmativas (en líneas generales) de los encuestados, cabe destacar mejoras y conclusiones para que QR-APDID se convierta en la mejor herramienta de apoyo e información que existe, tratando de definir correctamente sus objetivos y tratando de encontrar sus "puntos débiles" para conseguir que funcione a su máximo rendimiento.

6.1 Conclusiones

QR-APDID es una aplicación didáctica realizada en Java (J2ME) cuyo objetivo inicial era el de ser una aplicación para ayudar a encontrar lugares cercanos dentro de un campus universitario pero que, con el tiempo, la idea de información fue cambiando hasta evolucionar en la contextualización de ámbitos más comerciales o la búsqueda de destinos en zonas más amplias, convirtiendo a QR-APDID en una aplicación de ámbito más general, de información más específica y de una ayuda aun más grande para el usuario/administrador.

El contexto depende de cuál sea el uso que se le quiera dar porque a pesar de que el ámbito comercial pueda ser el más recurrido, también debemos conocer las posibilidades que este MIDlet ofrece dentro de un ámbito "abierto" en el que su principal objetivo es el de ayudar a llegar a un destino concreto mediante explicaciones y consejos para que el usuario conciba toda esta información y decida qué opción es la mejor para ir hacia el mejor lugar según sus prioridades.

Dentro del ámbito comercial, QR-APDID persigue distintos objetivos: esta aplicación quiere informar acerca de los productos, dando una explicación detallada del mismo (cosa que está al alcance de los administradores, que son los que determinan cuál es la información que quieren mostrar a sus clientes) y dando la posibilidad de que los clientes obtengan esta información de forma práctica, cómoda y segura. Pero, ¡atención!, el objetivo no es sustituir otros medios de información o suplantar posibles trabajos, si no el de completar estos tipos de información, conocer con exactitud lo que nos puede decir el fabricante acerca del producto, conocer ventajas e inconvenientes del mismo sin la necesidad de que alguien te lo diga de manera subjetiva. Poder corroborar si el precio es el adecuado o no para el producto, discernir en qué ocasiones, sitios o momentos este producto ejerce un mejor resultado, etc. Por todo esto, no podemos decir que QR-APDID suplante, si no que se trata de un "completador", de un "elemento de apoyo", de una herramienta que ayuda a un objetivo concreto.

Por otra parte, QR-APDID también es un "guía", algo con lo que poder encaminar nuestros pasos, pero esto no es solo en el ámbito comercial, sino que también se puede aplicar al ámbito "abierto". Es un lujo poder llegar a un sitio y poder ver las distintas posibilidades que se pueden tener debido a nuestra posición estratégica sin

necesidad de llevar un GPS o sin necesidad de conocer el lugar (el lugar puede ser cualquiera. Hemos hablado mucho de sitios abiertos como montañas o ríos, pero también debemos de englobar zonas urbanas, o incluso campus universitarios como era la idea inicial de todo este proyecto), simplemente haciendo una foto a un CQR y conociendo las posibilidades y caminos que se nos dan. Por tanto, QR-APDID nos guía, aparte de informar, para llegar a ese sitio que deseamos, sin perder de vista que esta herramienta también puede ser una ayudante de un GPS, por ejemplo, o de señales estáticas de información, como en el caso comercial, donde ya hemos repetido más de una vez que QR-APDID no suplanta, si no que complementa la información que se pueda obtener de otras maneras.

En conclusión, QR-APDID puede ser un elemento práctico, pero a la vez potente, de herramienta que nos permite desarrollar nuestros conocimientos de forma sencilla sin necesidad de depender de otros elementos de información (si es que no queremos depender de ellos). Busca la sencillez puesto que quiere orientarse a todas las edades de la población, para que todos podamos utilizar esta aplicación. Ya hemos dejado bien claro que el objetivo último es el de informar y guiar a las personas sea cual sea el ámbito, ya que esta aplicación tiene una gran relevancia si, dentro del contexto en el que nos encontramos, no tenemos los conocimientos necesarios para actuar o aún sabiendo queremos obtener información objetiva del entorno. Todo ello a nivel de usuario, pero a nivel de administrador hay que apuntar que esta herramienta sirve para incentivar el consumo (dentro de entornos comerciales) y agradar a los clientes (ya que siempre podrán acceder a la información necesaria); aparte de esto, el administrador puede conseguir una rápida y sencilla forma de guiar a los usuarios dentro de entornos naturales, urbanos, etc...

Ahora vamos a ver los “puntos críticos” de QR-APDID, mejorables desde el punto de vista de la persona que lo utiliza y de la persona que lo administra.

6.2 Mejoras

Antes de nada, tenemos que tener en cuenta que QR-APDID ha pasado por varias versiones y que, a pesar de todo, debería evolucionar a una versión más nueva para que este sea un producto de garantías y totalmente usable por parte de todos. Aún así, hay que reconocer que esta versión (QR-Apdid 1.0) tiene la gran funcionalidad que esta aplicación desea poseer pero, a continuación, vamos a explicar cuáles son los trabajos futuros para que esta aplicación obtenga la máxima funcionalidad y el menor número de errores posibles (además de que llegue a convertirse en un elemento usable para todo el mundo).

6.2.1 Acceso a cualquier tarjeta de memoria

En la versión 1.0 de QR-APDID el tipo de información de “file:///” es guardada en tarjetas MicroSD. La razón de que estas tarjetas sean las elegidas es por el hecho de que son las más usadas por los dispositivos móviles (un gran número de dispositivos móviles son compatibles con este tipo de tarjetas, que sirven como almacenamiento de datos), aparte de que se intenta eludir el problema que representa el hecho de que muchos terminales poseen una memoria interna muy minúscula. Esto está muy bien, puesto que se ha elegido el almacenamiento de datos más extendido en la industria, pero esto no quita que existan otros tipos de almacenamiento de datos en dispositivos móviles muy usados.

Una mejora que aumentaría mucho el “alcance” de QR-APDID sería el soporte de trabajar con el tipo de información procedente de las memorias sin distinción de las mismas, es decir, crear la forma en que cualquier usuario, sea cual sea su memoria, pueda utilizar su tarjeta de memoria con QR-APDID sin ningún tipo de restricción. Está clarísimo que aumentaría bastante el “alcance” puesto que cualquier persona con una tarjeta de memoria podría utilizar este tipo de información que QR-APDID proporciona (recordemos que el soporte de “file:// obtiene la información a partir de memorias, nunca de la memoria interna del terminal).

Esto se debería hacer de la siguiente manera: al iniciar QR-APDID, la aplicación debería hacer un chequeo del sistema de almacenamiento existente en el dispositivo en el cuál es inicializada. Una vez hecho este chequeo, QR-APDID ya sabrá de qué tipo de memoria se trata, por tanto podrá acceder a él sin ningún problema. Hay que constatar que el modelo de almacenamiento sería el mismo (a partir del directorio raíz de la tarjeta saldría la carpeta donde se guardan los archivos de QR-APDID) y que, por tanto, lo único que cambia es la sentencia de acceso. Aquí vienen algunos ejemplos de accesos (el último es el que usa QR-APDID 1.0):

* CFCard/ file:///CFCard/

* SDCard/ file:///SDCard/

* MemoryStick/ file:///MemoryStick/

* MicroSD/ file:///Mmc/

Aquí hemos visto algunos ejemplos de acceso a distintas tarjetas de almacenamiento. La solución a ello sería establecer una variable que guarde todos los tipos de accesos (como los vistos arriba) y mediante el chequeo de la tarjeta, nada más iniciar la aplicación, decidir cuál es el “path” del sistema de almacenamiento. Como ya dijimos antes, QR-APDID establece una arquitectura rígida, por lo que después de este camino tendremos, en su raíz, la carpeta de “QRA” donde se guardan los archivos necesitados para la interactuación del usuario con el entorno donde se encuentre.

Esta mejora sería clave para que QR-APDID sea utilizado por el máximo número de personas posibles pero habría más mejoras posibles como, por ejemplo, el establecimiento de servidor por parte de usuario, cliente (éste es el siguiente punto).

6.2.2 Establecimiento del servidor

En la primera versión de QR-APDID (no es la 1.0, si no la versión 0.1, que es como la versión inicial de la aplicación) la manera de conectarse al servidor era la misma que en todas las versiones existentes y por existir de esta aplicación, pero con una notable diferencia: la forma en que se obtiene la url del servidor. Este MIDlet obtiene la url del servidor a partir de una variable guardada como propiedad de la aplicación (en este caso, era una propiedad definida por el usuario). Esta opción (la inicial por mi parte para cambiar de manera sencilla la url del server siempre que fuese preciso) era la más eficiente, en principio, para poder disponer de un modelo sencillo de sincronización de QR-APDID con el servidor, pero hubo problemas. Estos problemas fueron apareciendo a medida que la aplicación era probada en distintos terminales móviles. En algunos terminales móviles funcionaba correctamente la conexión al servidor, pero en otros no. La causa de este problema era que la propiedad guardada como propiedad del sistema no era leída por parte del terminal, es decir, que era como si no se hubiese definido nada (en pruebas se veía que la variable que guardaba la url del server, en el hilo de ejecución, era una cadena vacía de caracteres). El motivo de este hecho es desconocido para mí, pero esto llevó a buscar otra manera de definición de la url del server.

En la 1.0, la url se instaura “grabada en la piel”, o sea, se pone a piñón en la clase principal de la aplicación. Evidentemente, de esta forma desaparece todo el problema de dispositivos móviles ya que en ningún momento se pierde la cadena en la que se describe la url del server. Por tanto, de esta forma se han probado multitud de dispositivos móviles con soporte de MIDP 2.0 y el resultado es correcto: la aplicación funciona correctamente sin ningún problema. Esta es la forma de adquirir la url del server por parte de QR-APDID, pero ha sido corregir o intentar subsanar un problema para tener otro: la restricción del servidor. Claramente se puede observar que el problema del servidor es importante ya que al ser “grabada a fuego” la url del mismo se obliga al administrador a poner un servidor con la misma url que QR-APDID contiene para interactuar con el servidor: 198.168.1.33. Esto no es para nada eficiente ya que el administrador tiene todo el derecho del mundo a poner la url que desee o incluso dejar que mediante DHCP se obtenga una url dinámica dentro de un entorno local (no hay que perder de vista que la aplicación la obtienen los usuarios, por tanto si ésta contiene dicha url grabada, esto significa que el administrador ó, en su caso, el server, debe tener la misma url para poder ser solicitada).

La idea de mejora en este aspecto es la siguiente: justo al inicializar la aplicación, debería de aparecer la típica pantalla explicativa de cómo utilizar QR-APDID. Después de esto, se debería ejecutar un proceso que empieza con la aparición de otra pantalla en la que pide que se introduzca la ip del servidor (una pantalla con un mensaje: “¿IP del server?”); en el espacio en blanco se pone la dirección si hay server

en el contexto de utilización, y si no se pone un 0. Con esta información, QR-APDID establecerá la dirección IP del server con el que se va a interactuar en esta sesión (y en el contexto en el que se encuentra el usuario/cliente), y sabrá si no hay ningún server con el que interactuar si el usuario/cliente ha puesto un 0.

De esta manera el uso de servidores es muy dinámico y se consigue una libertad total por parte del administrador que solamente debe ofrecer la IP del server a usuarios/clientes para la utilización del mismo.

Esta última es otra mejora importante que, además, debería de ser tratada antes de su teórica puesta en el mercado. Por ello podríamos decir que es incluso más importante que la primera.

Ahora vamos a ver una tercera y última mejora que se podría hacer a QR-APDID. Esta ya no tiene tanto que ver con la aplicación en si (es decir, con el código en concreto) si no con los CQR y su codificación.

6.2.3 Tamaño de CQR

Debido a la restricción que puede provocar la cámara del terminal móvil con el que se esté usando QR-APDID, se debe tener en cuenta el tamaño de la codificación del CQR. Un claro ejemplo de esto es que se puede tener una cámara de poca resolución y, por tanto, no poder capturar toda la información contenida de un CQR. Este hecho lleva a una mala descodificación del mismo (en la misma no se establece toda la información que se necesita para la descodificación) y, como consecuencia, al mal funcionamiento de QR-APDID al no poder descodificar los CQR.

Esto no se trata de código java, o de clases, o la estructura de la aplicación. Esto tiene más que ver con el diseño del ámbito que el administrador está dispuesto a ofrecer a sus clientes/usuarios. Este diseño debe ser accesible a todas estas personas ya que el buen funcionamiento de QR-APDID no solo depende de sí mismo, si no que el terminal puede representar un incordio (con esto pasa lo mismo que con la memoria interna del teléfono, cosa que es subsanada mediante la utilización de tarjetas de memoria, cosa que, a su vez, se sabe que no todos los terminales disponen de una) y esto hay que tenerlo en cuenta. El diseño del contexto va a cargo del administrador, que es quien decide qué cosas poner para que los usuarios/clientes interactúen con ellas, además de saber qué elementos o informaciones deben ser mostrados. Para ello se pide que el diseño sea sencillo, que se eviten multitud de caracteres de información codificada, ya que las cámaras de los móviles son muy variables (las hay muy buenas y las hay no tan buenas). El objetivo principal de QR-APDID es informar y guiar al mayor número de personas posibles, por tanto esta restricción se debe tener muy en cuenta a la hora del diseño porque la cámara puede ser un elemento restrictivo primordial.

Ya hemos visto todos los posibles trabajos futuros, las posibles mejoras que se le pueden hacer a QR-APDID. Todas estas mejoras permitirán a QR-APDID convertirse en una potente, pero a la vez sencilla, herramienta de consulta de información. Como hemos visto, dos de las tres mejoras son de codificación, mientras que la última es de diseño. Estas mejoras de codificación deberían de ser reales antes de que la aplicación empezara a distribuirse y a ser utilizada por las personas. La mejora de diseño ya es algo totalmente dependiente del administrador del ámbito concreto por lo que esto siempre va a ser algo dependiente de las personas y no de la aplicación.

Llegados a este punto, cabe destacar que dichas mejoras son realizables y que su único objetivo es el de contribuir a la mejor utilización de la aplicación para que el fin perseguido (el puramente didáctico) sea conseguido.

6.2.4 Mejoras en la parte Web

Hay dos claras mejoras que se pueden hacer a esta aplicación web:

La primera es totalmente estética, ya que lo que se busca con la interfaz gráfica no es más que la sencillez y la efectividad de recoger estadísticas y la creación de códigos QR, pero no ha habido concentración en la parte puramente estética y representativa, por lo que una mejora sería dar estilo a esta página para que quede “más bonita” de cara a aquéllos que usen la aplicación.

La segunda correspone con la gestión de estadísticos. Se tiene una clase en la que se mantienen ciertas variables que guardan el valor de las veces que se utiliza la aplicación, pero esto no es del todo eficiente, ya que si el servidor cae, estos datos se pueden perder, así que sería más interesante contener un archivo txt donde se vayan escribiendo los valores correspondientes a los contadores. De esta manera, siempre leeremos del archivo y, por tanto, puede ocurrir cualquier cosa, que los estadísticos no se pederían (Al menos, los datos obtenidos hasta el momento en el que hubiese una posible caída del servidor).

A continuación vienen los costos de realizar este proyecto.

Capítulo 7: Presupuesto

Aquí aparecen los costes del proyecto. Son una aproximación, así que deben ser tomados como tal:

- Luz en mi casa:

MES	Días a la semana	Semanas al mes	Horas dedicadas mensuales	Gasto de luz
Octubre	5	5	100	12,5
Noviembre	5	4	80	10
Diciembre	4	3	48	6
Enero	2	1	8	1
Febrero	2	1	8	1
Marzo	4	4	64	8
Abril	5	4	80	10
Mayo	2	2	16	2
Junio	3	3	36	4,5

TOTAL: 55

Gasto en luz de 55 euros en mi casa.

- Luz fuera de mi casa

MES	Días a la semana	Semanas al mes	Horas dedicadas mensuales	Gasto de luz
Octubre	2	2	12	1,5
Noviembre	1	2	6	0,75
Diciembre	2	1	6	0,75
Enero	3	1	9	1,125
Febrero	3	1	9	1,125
Marzo	2	2	12	1,5
Abril	1	3	9	1,125
Mayo	2	2	12	1,5
Junio	2	1	6	0,75

TOTAL:
10,125

Gasto de luz de 10,125 euros fuera de mi casa, en otras casas.

A los gastos de luz tanto fuera como dentro de casa no sumo los gastos de luz ocasionados en la universidad ya que, a la vez que es algo que pagan ellos, tampoco se de cuánto dinero se trata y de si la tarifa de pago es la misma que para particulares.

Ahora nos encargamos de temas de comidas fuera de casa (tanto en cafeterías como en casas ajenas):

- Comidas

MES	Veces que comí fuera	Precio medio de comidas	TOTAL euros mes
Octubre	4	7	28
Noviembre	2	7	14
Diciembre	2	7	14
Enero	3	7	21
Febrero	3	7	21
Marzo	4	7	28
Abril	3	7	21
Mayo	4	7	28
Junio	2	7	14

TOTAL: 189 Euros

Obtenemos un gasto total en comidas de 189 euros.

Ahora vamos a hacer una estimación de lo gastado en transporte público, ya que he tenido que hacer desplazamientos en muchas ocasiones. Todos estos precios están bastante estimados:

- Transporte público

MES	Número de viajes	Precio medio de viaje	TOTAL euros mes
Octubre	14	2,5	35
Noviembre	16	2,5	40
Diciembre	14	2,5	35
Enero	8	3	24
Febrero	8	3	24
Marzo	12	3	36
Abril	18	3	54
Mayo	6	3	18
Junio	14	3	42

TOTAL: 308 Euros

Los viajes en transporte público hacen un coste total de 308 euros.

Ahora vamos a estimar lo gastado en viajes en coches particulares (está estimado):

- Gasolina particulares

MES	Km recorridos en mes	Precio medio gasolina	TOTAL euros mes
Octubre	50	1,01	50,5
Noviembre	48	1,2	57,6
Diciembre	20	1,2	24
Enero	0	1,15	0
Febrero	0	1,4	0
Marzo	23	1,3	29,9
Abril	50	1,2	60
Mayo	10	1,3	13
Junio	30	1,2	36

TOTAL: 271 Euros

El total en gasolinas particulares es de 271 euros.

Por lo tanto, haciendo una suma de todos los gastos, nos sale un total de: 833.125 euros. Este es el resultado total, a parte de todos los posibles gastos correspondientes con el trabajo que significa desarrollar el proyecto y energía consumidos. A estos gastos debemos sumar los gastos indirectos, como son los del uso de mesas, sillas, papel, uso de instalaciones, etcétera… Estimamos que esto resulta ser un 25% de lo que se gasta de costes directos, por lo que el total de dinero gastado asciende a 1041.41 euros.

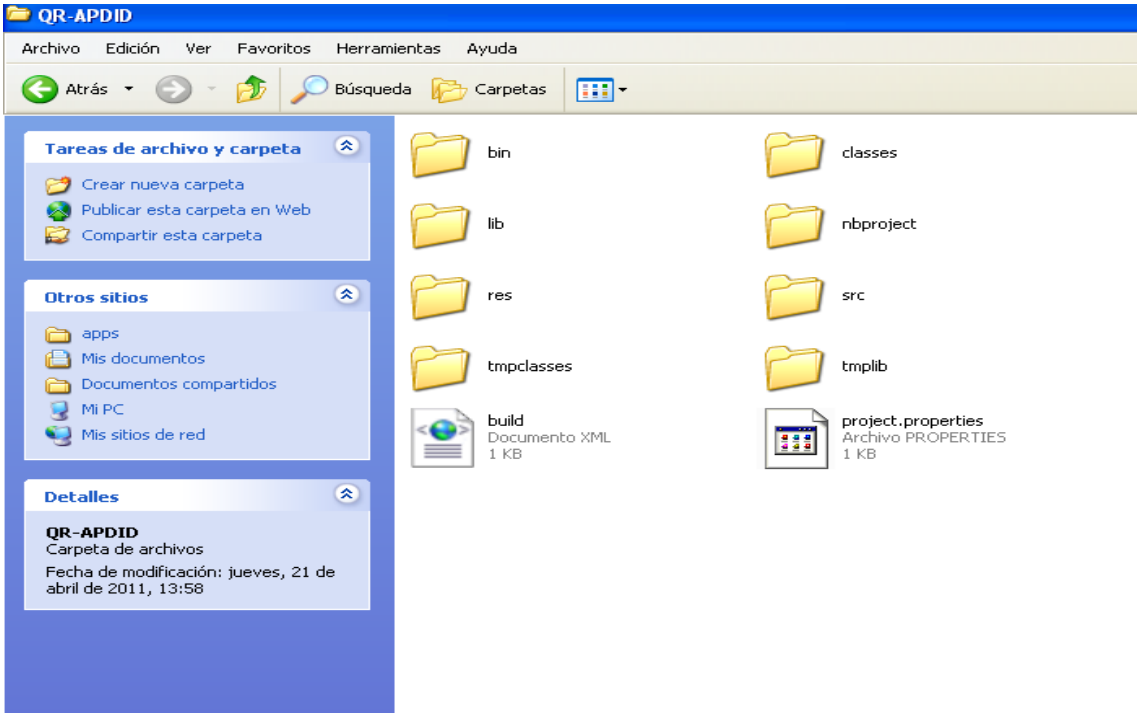
De todos modos, a todo esto se le deberían sumar gastos posteriores al verano, resultantes de las correcciones realizadas en la memoria, hechos que también pertenecen al desarrollo final de todo el proyecto y el valor incalculable del tiempo tardado en la realización del mismo, algo que no tiene valor neto, si no que tiene una simbología importantísima en toda persona.

Anexo I

En este anexo viene explicado cómo hacer que QR-APDID funcione y cómo instalar dicha aplicación en su terminal móvil, a la vez que la creación de los CQR:

Lo primero de todo, hay que instalar el Samsung SDK for Java o el Wireless Toolkit for J2ME (el segundo es el preferible ya que es más general. En mi caso usé el primero puesto que el móvil con el que más pruebas hice es el mío, que es un Samsung), ya que esta aplicación nos permitirá establecer un entorno java en el que podremos trabajar con clases y paquetes. Como las clases ya están totalmente definidas y establecidas, lo único que se debe hacer es realizar un empaquetado de las clases para obtener un fichero ".jar", pero antes veremos algunas cosas interesantes que se deben hacer.

Tenemos que establecer los archivos necesarios en las carpetas correspondientes de la aplicación. Con las clases ya creadas, hay que crear un proyecto en la aplicación para poder tener el entorno de trabajo. Esto permitirá obtener un directorio de la aplicación con una serie de carpetas que ya hemos explicado antes en otro capítulo de esta memoria. Para QR-APDID será semejante al siguiente:



En "src" tenemos los ".java", en "res" debemos colocar las imágenes de nuestra aplicación (archivos ".png"). Todas las demás carpetas poseen archivos generados automáticamente al trabajar con la aplicación, por tanto esta es la forma en la que hay que colocar los archivos para que la aplicación funcione correctamente.

Tras todo esto, procedemos al empaquetado en el menú de la aplicación. En la carpeta de “bin” aparecerá el archivo “QRAPDID.jar” que es el que nos tenemos que llevar al móvil e instalar siguiendo los pasos contados.

El servidor se debe establecer como dijimos en el capítulo donde se habla de Apache-Tomcat (puesto que este es el elemento que va a contener nuestro propio servidor de archivos) y debemos establecer el fichero “.class” del servlet dentro de la carpeta con destino WEB-INF/classes (el fichero “.java” no nos interesa y, además, se debe configurar el “web.xml” como también vimos antes).

Los archivos que se quieran obtener de tipo “file” sencillamente deben introducirse en la memoria del móvil previamente, al igual que pasa con los archivos de tipo “url”, solo que estos deben meterse en el server.

Los CQR se deben generar a partir de páginas de internet y deben imprimirse en papel (como norma general) para que se puedan descodificar correctamente, sin manipular el tamaño de los mismos. Los tamaños varían, por lo que hay que tener en cuenta lo dicho antes para que no haya problemas de descodificación.

Con estos sencillos pasos se pueden obtener unos grandes resultados y un correcto funcionamiento de la aplicación. No hay que olvidar que los pasos detallados de configuración y funcionamiento han sido tratados y desarrollados en capítulos anteriores, por lo que en este último capítulo se da un ligero esbozo de cómo conseguir hacer funcionar todo el entorno de QR-APDID.

Anexo II

Los directorios en el servidor:

Para el correcto funcionamiento de QR-APDID para tratar la información alojada en el servidor se necesitan establecer definiciones inamovibles para los caminos a los elementos alojados en el mismo. Se hace una distinción entre los distintos tipos de información para que sea fácil para el administrador alojar los archivos y hacer una distinción sencilla.

Dentro de la raíz del directorio en el servidor, al cual le llamamos MYSERVER (Este es el directorio de nuestro servidor dentro de Apache-Tomcat, dentro de la carpeta de "webapps") aparece el directorio "WEB-INF", donde en él se alojan las clases de la aplicación web, que en este caso es solamente el "MyServlet.class" (dentro de la carpeta "classes") y el web.xml:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
 Licensed to the Apache Software Foundation (ASF) under one or more
  contributor license agreements.  See the NOTICE file distributed with
  this work for additional information regarding copyright ownership.
  The ASF licenses this file to You under the Apache License, Version 2.0
  (the "License"); you may not use this file except in compliance with
  the License.  You may obtain a copy of the License at
      http://www.apache.org/licenses/LICENSE-2.0
  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
->
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  version="2.5">
 <display-name>Server de recursos remotos</display-name>
 <description>
   A scriptable management web application for the Tomcat Web Server;
   Manager lets you view, load/unload/etc particular web applications.
 </description>
 <servlet>
   <servlet-name>MyServlet</servlet-name>
   <servlet-class>MyServlet</servlet-class>
 </servlet>
 <!-- Define the Manager Servlet Mapping -->
 <servlet-mapping>
   <servlet-name>MyServlet</servlet-name>
     <url-pattern>/servl</url-pattern>
 </servlet-mapping>
</web-app>
```

A la altura de “WEB-INF” (dentro del espacio de directorios que estamos estudiando) se encuentran los directorios donde se guardan los archivos:

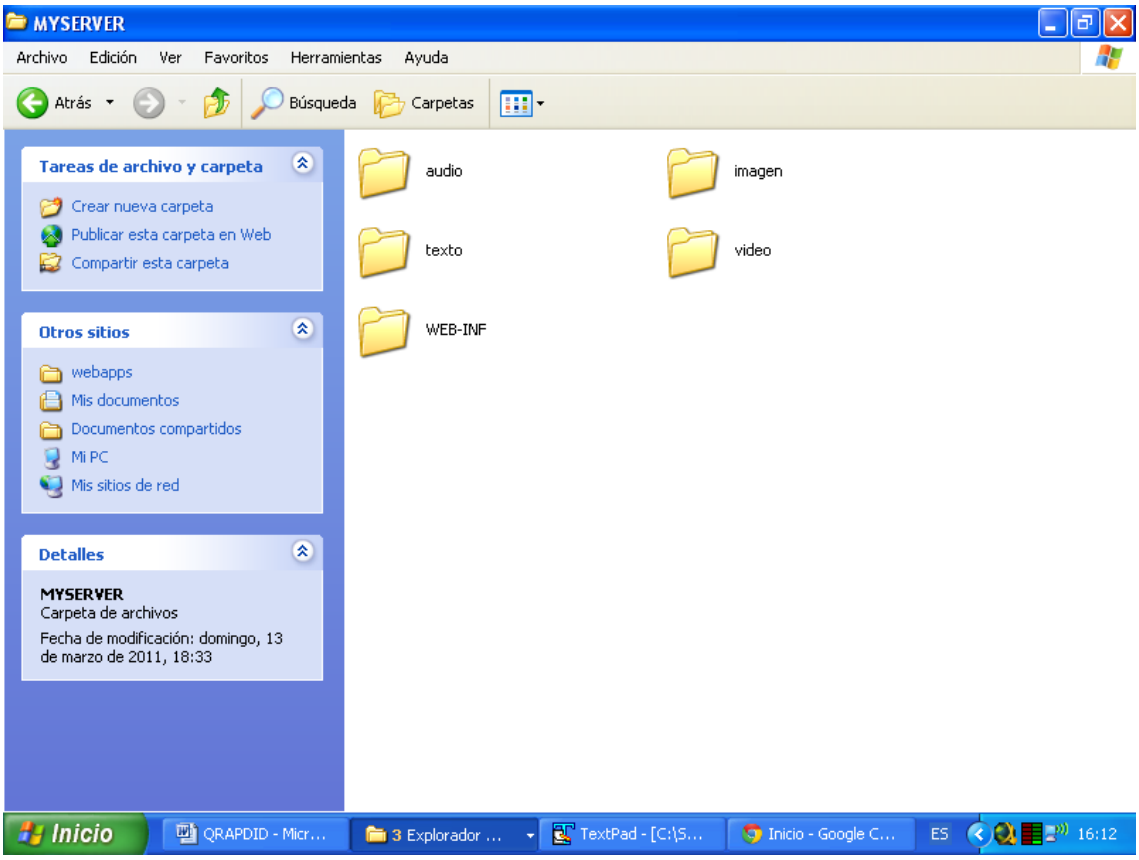
TEXTO – la carpeta se llama “texto” y dentro de ella se alojan todos los archivos de tipo texto (“.txt”).

IMAGEN – la carpeta e llama “imagen” y en lla se encuentran los archivos “.png”.

AUDIO – Misma idea que los anteriores con archivos de audio con extensión “.mp3” guardados en una carpeta llamada “audio”.

VIDEO – Como todos los anteriores, pero con archivos “.mp4” guardados en una carpeta llamada “video”.

Con esta arquitectura conseguimos dos cosas: que haya un estándar para el almacenamiento de los archivos en el servidor y que sea una manera bastante sencilla ya que, como podemos comprobar, no existe ninguna pérdida para alojar los archivos en el lugar que corresponda.



Ya queda definido todo lo referente al servidor, cuyo trato se debe hacer teniendo en cuenta los pasos definidos en el apartado de Apache-Tomcat para arranque del mismo y mantenimiento.

Anexo III

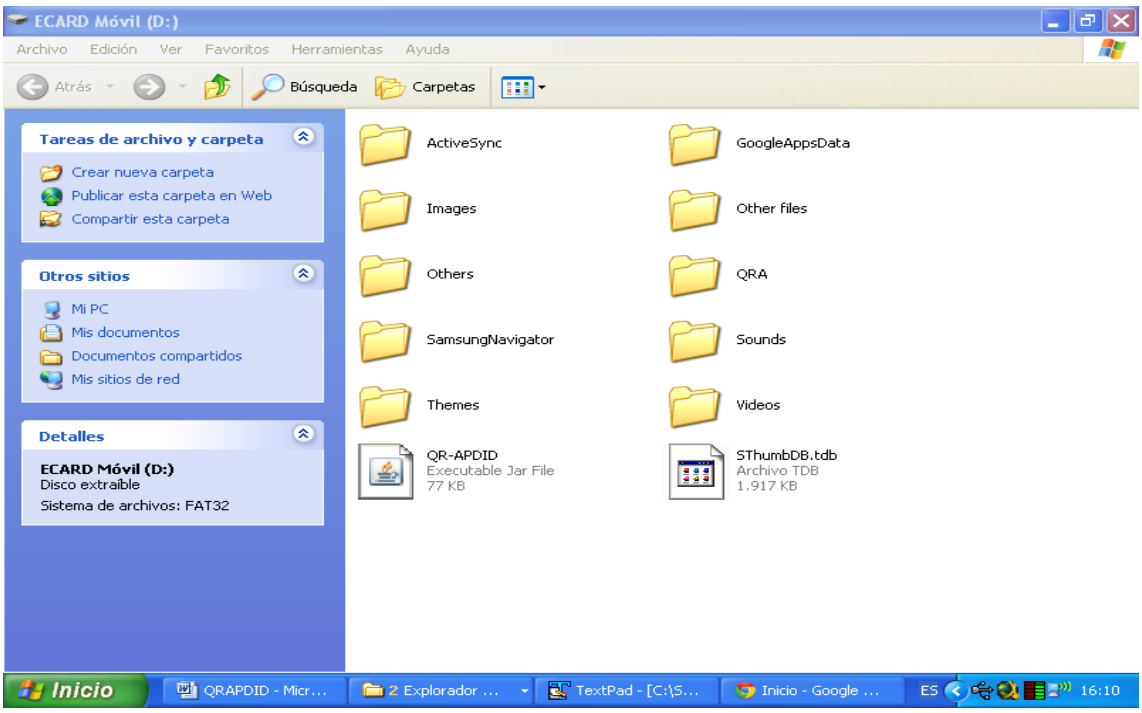
Los directorios en MicroSD

Las tarjetas **microSD** o**Transflash** corresponden a un formato de tarjeta de memoria flash más pequeña que la MiniSD, desarrollada por SanDisk; adoptada por la Asociación de Tarjetas SD bajo el nombre de «microSD» en julio de 2005. Mide tan solo 15 × 11 × 1 milímetros, lo cual le da un área de 165 mm². Esto es tres veces y media más pequeña que la miniSD, que era hasta la aparición de las microSD el formato más pequeño de tarjetas SD, y es alrededor de un décimo del volumen de una SD card. Sus tasas de Transferencia no son muy altas, sin embargo, empresas como SanDisk han trabajado en ello, llegando a versiones que soportan velocidades de lectura de hasta 10 Mbit/s.

Debido a que su coste como poco duplica el de una Secure Digital equivalente, su uso se ciñe a aplicaciones donde el tamaño es crítico, como los teléfonos móviles, sistemas GPS o tarjetas Flash para consolas de mano (como GameBoy Advance,Nintendo DS o Nintendo DSi).

Debido a que su coste como poco duplica el de una Secure Digitalequivalente, su uso se ciñe a aplicaciones donde el tamaño es crítico, como los teléfonos móviles, sistemas GPS o tarjetas Flash para consolas de mano (como GameBoy Advance, Nintendo DS o Nintendo DSi). ([1]).

Aquí sí que hay cambios con respecto a la forma de guardar en el server los distintos tipos de archivos. De hecho, aquí no hay ninguna distinción, simplemente una carpeta en la que se meten todos los archivos necesarios para que QR-APDID interactúe con ellos, de tal forma que se sirven todos los archivos al mismo nivel sin necesidad de utilizar otro tipo de forma de visualización de la información. En definitiva, que un usuario que quiera tener la posibilidad de que QR-APDID acceda a los archivos guardados en su terminal, simplemente debe crear una carpeta (en el directorio raíz de la MicroSD) llamada QRA. En esta carpeta se meterán, sin excepción, todos los archivos que se vayan a necesitar para interactuar en un entorno determinado:



El directorio al que se accede cuando se establece una conexión con la MicroSD es (también entendido como url a la que se accede, o uri):

file:///Mmc/QRA/*

Donde "*" es el nombre del archivo que el CQR indica en su codificación. Mmc es el nombre del directorio con el que se accede a la MicroSD y, como ya vimos antes, se establece una carpeta donde se guardan todos los archivos necesarios, esa carpeta es "QRA" (este es el nombre definido para estandarizar el acceso a la memoria del terminal por parte de la aplicación, por lo que debe ser esta la forma y no otra en que se nombre a dicha carpeta).

Por último, para ver la gran inmersión de MicroSD en el mercado, vamos a ver una lista de ejemplo (de hace 4 años, para que se vea que es una tecnología totalmente asentada en el mercado, puesto que ahora mismo el uso de las MicroSD es total) de móviles que soportan MicroSD ([11]):

LG Chocolate 3G (KU800)
LG KU311
LG KU450
LG L600V
LG PRADA
LG SHINE
Motorola Z3
Motorola Z8
Motorola A780
Motorola E770
Motorola L7
Motorola K1
Motorola V1100
Motorola V360
Motorola V360i

Motorola V3x
Motorola v635
Motorola W510
Nokia 5200
Nokia 5300
Nokia 5700
Nokia 6085
Nokia 6086
Nokia 6100 Navigator
Nokia 6111
Nokia 6131
Nokia 6136
Nokia 6151
Nokia 6234
Nokia 6234 David bisbal
Nokia 6280
Nokia 6288
Nokia 6300
Nokia 7300
Nokia 7373
Nokia 7390
Nokia N76
Nokia N80
Nokia N95
Samsung E250
Samsung E840 Silver
Samsung U600
Samsung Z240
Samsung ZV40
Samsung ZV60
Sharp 770SH
Sharp 770SH MC LAREN
Toshiba TS605
Toshiba TS705

Bibliografía

-[1] WIKIPEDIA, la enciclopedia libre. Para definiciones varias y explicaciones sobre diversas tecnologías (los códigos QR, por ejemplo).
-[2] WIKIPEDIA, the free enciclopedy. Igual que la línea anterior pero buscando la información en lengua inglesa.
-[3] MIDP Profile de "java.sun". Especificaciones de la tecnología J2ME y, en particular, de todo lo relacionado con el perfil MIDP 2.0 editado por SUN Microsistems.
-[4] Apuntes de Software de Comunicaciones, parte referida a J2ME (profesora: Florina Almenárez Mendoza). Los dibujos de ciclo de vida y otras explicaciones muy prácticas para el tema que nos compete.
-[5] Aplicación QRMIDlet sacada de:
http://www.google.com/codesearch/p?hl=es#GrRU2R20Gj4/trunk/QRMidlet/src/DecodeThread.java&q=qrmidlet&d=3
-[6] Diversas páginas de documentación sobre el mundo de la Realidad Aumentada.
-[7] Web de Softonic. Explicación de entornos de programas de desarrollo.
-[8] Manual de prácticas de Ingeniería Telemática de la Universidad Carlos III de Madrid. Explicación detallada de la configuración de Apache Tomcat o Eclipse.
-[9] Taringa.net : Algunas fotos del proyecto han sido sacados de esta página web.
-[10] Tutorial de Wireless Toolkit.
- http://jcalderon.wordpress.com para post referente a la instalación de Tomcat en XP.
-[11] guia.mercadolibre.com. Lista de móviles que soportan MicroSD
-[12] http://eduarea.wordpress.com. Artículo sobre las ventajas de los códigos QR

-[13] Manual de Apache-Tomcat